



**Universidade de Aveiro**  
2017

Departamento de Eletrónica, Telecomunicações e  
Informática

**José Horácio Fradique  
Duarte**

**A Framework for the Management of Deformable  
Moving Objects**

**Um *Framework* para Manipulação de Objetos Móveis  
Deformáveis**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor José Manuel Matos Moreira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Paulo Miguel de Jesus Dias, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro



## **O júri /The jury**

Presidente / President

Prof. Doutor Joaquim Arnaldo Carvalho Martins  
Professor Catedrático da Universidade de Aveiro

## **Vogais / Examiners**

Committee

Prof. Doutor Alexandre Miguel Barbosa Valle de Carvalho  
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor José Manuel Matos Moreira  
Professor Auxiliar da Universidade de Aveiro (Orientador)

Prof. Doutor Paulo Miguel de Jesus Dias  
Professor Auxiliar da Universidade de Aveiro (Co-orientador)



**Agradecimentos /  
Acknowledgments**

Quero agradecer aos meus orientadores, Professor José Moreira e Professor Paulo Dias, pela sua disponibilidade, pela sua ajuda e pelo bom relacionamento que sempre tivemos durante esta aventura.

Quero agradecer à Professora Enide Martins pela sua disponibilidade e pelo seu tempo para discutirmos algumas questões mais matemáticas.



## Keywords

Moving Objects, Spatiotemporal Data Models, Spatiotemporal Databases, Compatible Triangulation, Rigid Interpolation.

## Abstract

There is an emergence of a growing number of applications and services based on spatiotemporal data in the most diverse areas of knowledge and human activity. The Internet of Things (IoT), the emergence of technologies that make it possible to collect information about the evolution of real world phenomena and the widespread use of devices that can use the Global Positioning System (GPS), such as smartphones and navigation systems, suggest that the volume and value of these data will increase significantly in the future. It is necessary to develop tools capable of extracting knowledge from these data and for this it is necessary to manage them: represent, manipulate, analyze and store, in an efficient way. But this data can be complex, its management is not trivial and there is not yet a complete system capable of performing this task.

Works on moving points, that represent the position of objects over time, are frequent in the literature. On the contrary there are much less solutions for the representation of moving regions, that represent the continuous changes in position, shape and extent of objects over time, e.g., storms, fires and icebergs. The representation of the evolution of moving regions is complex and requires the use of more elaborate techniques, e.g., morphing and interpolation techniques, capable of producing realistic and geometrically valid representations.

In this dissertation we present and propose a data model for moving objects (moving points and moving regions), in particular for moving regions, based on the concept of mesh and compatible triangulation and rigid interpolation methods. This model was implemented in a framework that is not client or application dependent and we also implemented a spatiotemporal extension for PostgreSQL that uses this framework to manipulate and analyze moving objects, as a proof of concept that our framework works with real applications. The tests' results using real data, obtained from satellite images of the evolution of 2 icebergs over time, show that our data model works. Besides the results obtained one important contribution of this work is the development of a basic framework for moving objects that can be used as a basis for further investigation in this area. A few problems still remain that must be further studied and analyzed, in particular, the ones that were found when using the compatible triangulation and rigid interpolation methods with real data.





## Palavras-Chave

Objetos Móveis, Modelos de Dados Espaço-temporais, Bases de Dados Espaço-temporais, Triangulação Compatível, Interpolação Rígida.

## Resumo

Assistimos ao aparecimento de um número crescente de aplicações e serviços baseados em dados espaço-temporais nas mais diversas áreas do conhecimento e da atividade humana. A internet das coisas (IoT), o aparecimento de novas tecnologias que permitem obter dados sobre a evolução de fenómenos do mundo real e o uso generalizado de dispositivos que usam o sistema de posicionamento global (GPS), por exemplo, *smartphones* e sistemas de navegação, sugerem que o volume e o valor destes dados aumente significativamente no futuro. Torna-se necessário desenvolver ferramentas capazes de extrair conhecimento destes dados e para isso é necessário geri-los: representar, manipular, analisar e armazenar, de uma forma eficiente. Mas estes dados podem ser complexos, a sua gestão não é trivial e ainda não existe um sistema completo capaz de executar essa tarefa.

Existe muito trabalho na literatura sobre pontos móveis, que representam as alterações da posição de objectos ao longo do tempo, mas existe muito menos trabalho realizado sobre regiões móveis, que representam as alterações da posição e da forma de regiões ao longo do tempo, por exemplo, uma tempestade, um incêndio ou um derramamento de petróleo. A representação da evolução de regiões móveis ao longo do tempo é complexa e exige o uso de técnicas mais elaboradas, por exemplo, técnicas de *morphing* e interpolação, capazes de produzir representações realistas e geometricamente válidas.

Nesta dissertação apresentamos e propomos um modelo de dados para trabalhar com objetos móveis (pontos móveis e regiões móveis), em particular regiões móveis, baseado no conceito de malha e em métodos de triangulação compatível e interpolação rígida. Este modelo foi implementado num *framework* que é independente do cliente e da aplicação. Também implementámos uma extensão espaço-temporal para o sistema de gestão de base de dados PostgreSQL, que usa este *framework* para manipular e analisar objectos móveis, como uma prova de conceito que o nosso *framework* funciona com aplicações reais. Os resultados dos testes com dados reais, obtidos a partir de imagens de satélite da evolução de 2 icebergs ao longo do tempo, demonstram que o nosso modelo funciona. Para além dos resultados obtidos, um contributo importante desta dissertação é o desenvolvimento de um *framework* que pode ser usado como a base para trabalho futuro e investigação nesta área. Existem alguns problemas ainda por resolver e que devem ser analisados e estudados com mais cuidado, em particular, os que foram encontrados quando usámos os métodos de triangulação compatível e interpolação rígida em dados reais.



---

## Contents

|  |    |
|--|----|
| 1. Introduction.....   | 1  |
| 1.1 Context, Motivation and Goals.....                       | 2  |
| 1.2 Outline of this Dissertation.....                        | 3  |
| 2. Representation and Management of Spatiotemporal Data..... | 5  |
| 2.1 Concepts .....   | 5  |
| 2.2 Spatiotemporal Databases.....                            | 6  |
| 2.2.1 Applications .....                                     | 7  |
| 2.2.2 Architecture.....                                      | 7  |
| 2.2.3 Data Models and Operations .....                       | 8  |
| 2.2.4 Spatiotemporal Query Languages .....                   | 11 |
| 2.3 Spatiotemporal ORDBMSs.....                              | 12 |
| 2.4 Representation of Moving Regions.....                    | 15 |
| 2.5 Summary.....   | 17 |
| 3. A Discrete Data Model for Moving Objects .....            | 19 |
| 3.1 Overview.....  | 22 |
| 3.2 Notation and Assumptions .....                           | 23 |
| 3.3 Type System Data Types .....                             | 24 |
| 3.3.1 Base Types and Time Types .....                        | 24 |
| 3.3.2 Interval and Period Types .....                        | 24 |
| 3.3.3 Spatial types .....                                    | 24 |
| 3.3.4 Sliced Representation for Moving Objects .....         | 26 |
| 3.3.5 Temporal Units for Base Data Types.....                | 27 |
| 3.3.6 Temporal Units for Spatial Data Types .....            | 27 |
| 3.3.7 Moving Types .....                                     | 28 |
| 3.4 Operations on Moving Types.....                          | 28 |
| 3.4.1 Predicates .....                                       | 29 |

|  |    |
|--|----|
| 3.4.2 Set Operations .....   | 29 |
| 3.4.3 Numeric .....  | 30 |
| 3.4.4 Projection to Domain and Range .....                           | 30 |
| 3.4.5 Interaction with Domain and Range .....                        | 30 |
| 3.4.6 Constructors .....   | 31 |
| 3.5 Summary .....  | 31 |
| 4. Implementation .....  | 33 |
| 4.1 Overview .....   | 33 |
| 4.2 Technologies and Architectures .....                             | 36 |
| 4.2.1 To Implement a Framework for Moving Objects .....              | 37 |
| 4.2.2 To implement a Spatiotemporal Database Extension .....         | 40 |
| 4.3 SPTMesh - A Framework for Moving Objects .....                   | 40 |
| 4.3.1 Dependencies .....   | 40 |
| 4.3.2 Data Structures .....  | 41 |
| 4.3.3 Operations .....   | 43 |
| 4.3.4 Triangulation, Smoothing and Interpolation Methods .....       | 46 |
| 4.3.5 Continuity for Unit Types .....                                | 47 |
| 4.3.6 A Spatiotemporal Well-Known Text Form for Moving Objects ..... | 47 |
| 4.3.7 Architecture .....   | 49 |
| 4.3.8 Issues .....   | 50 |
| 4.3.9 Usage and Installation .....                                   | 52 |
| 4.4 MeshGIS - A Spatiotemporal Extension for PostgreSQL .....        | 53 |
| 4.4.1 Data Structures .....  | 54 |
| 4.4.2 Operations .....   | 57 |
| 4.4.3 Architecture .....   | 59 |
| 4.4.4 Usage and Installation .....                                   | 60 |
| 4.5 A Framework for Future Work and Investigation .....              | 61 |
| 4.5.1 Code Structure .....   | 61 |
| 4.5.2 Extending the Framework .....                                  | 63 |
| 4.6 Summary .....  | 64 |

|  |    |
|--|----|
| 5. Data Model Evaluation .....                           | 65 |
| 5.1 Datasets.....  | 65 |
| 5.2 Tests.....   | 65 |
| 5.3 Summary.....   | 72 |
| 6. Discussion.....                                       | 75 |
| 6.1 Compatible Triangulation .....                       | 75 |
| 6.2 Interpolation.....                                   | 77 |
| 6.2.1 Degenerated Triangles and Invalid Geometries ..... | 77 |
| 6.2.2 Unwrap Method .....                                | 79 |
| 6.2.3 Smoothing Method Relevance .....                   | 80 |
| 6.3 Smoothing Method Performance .....                   | 81 |
| 6.4 Continuity .....                                     | 82 |
| 6.4.1 Mesh Objects .....                                 | 82 |
| 6.4.2 Unit Real Objects.....                             | 83 |
| 6.5 Other Issues .....                                   | 83 |
| 6.6 Summary.....   | 84 |
| 7. Conclusions and Future Work .....                     | 85 |
| 7.1 Conclusions.....                                     | 85 |
| 7.2 Main Contributions .....                             | 86 |
| 7.3 Future Work.....                                     | 87 |
| 8. Bibliography .....                                    | 89 |



---

## List of Figures

|   |    |
|---|----|
| Figure 1.1 - 2 observations of the evolution of an iceberg over time. ....  | 2  |
| Figure 2.1 - Sliced representation of: a moving real (left) and 2 moving points (right). Source: (Forlizzi et al., 2000)..... | 10 |
| Figure 2.2 - A Moving point with various types of movement. Source: (Nikos Pelekis & Theodoridis, 2007). ....                 | 10 |
| Figure 2.3 - Observations of an iceberg evolving continuously over time. ....   | 15 |
| Figure 2.4 - Examples of possible interpolation results that represent the region's evolution during a slice. .               | 16 |
| Figure 2.5 - Two different morphing techniques. Source: image adapted form (Baxter et al., 2008). ....                        | 17 |
| Figure 3.1 - 2 observations of the evolution of an iceberg over time. ....  | 19 |
| Figure 3.2 - A geometry with collinear points. ....   | 20 |
| Figure 3.3 - 2 meshes. ....   | 20 |
| Figure 3.4 - Steiner points. ....   | 21 |
| Figure 3.5 - A mesh obtained not using (left) and using (right) the smoothing method. ....                                    | 21 |
| Figure 3.6 - A mesh of an iceberg, obtained using our framework for moving objects. ....                                      | 24 |
| Figure 4.1 - Overall overview of the architecture of a system that uses MeshGIS and SPTMesh. ....                             | 36 |
| Figure 4.2 - SPTMesh architecture using C++ and GEOS. ....  | 37 |
| Figure 4.3 - SPTMesh architecture using Python and Shapely. ....  | 38 |
| Figure 4.4 - SPTMesh architecture using PostGIS. ....   | 38 |
| Figure 4.5 - GEOS Architecture. ....  | 39 |
| Figure 4.6 - Interpolation of a geometry that coils during an interval. ....  | 47 |
| Figure 4.7 - SPTMesh architecture and available APIs. ....  | 49 |
| Figure 4.8 - The GEOS type system. Source: GEOS documentation. ....   | 49 |
| Figure 4.9 - The SPTMesh type system. ....  | 50 |
| Figure 4.10 - SPTMesh interpolation method implementation classes and decoupling infrastructure. ....                         | 51 |
| Figure 4.11 - A type system with a <i>mobject</i> and <i>uobject</i> types. ....  | 52 |
| Figure 4.12 - MeshGIS data structures relationships simplified diagram overview. ....   | 56 |
| Figure 4.13 - MeshGIS high level architecture overview. ....  | 60 |
| Figure 4.14 - PostGIS high level architecture overview with its external dependencies. ....                                   | 60 |
| Figure 4.15 - <i>Moving</i> types available in PostgreSQL after installing MeshGIS. ....                                      | 61 |
| Figure 5.1 - Coil interpolation test. ....  | 65 |
| Figure 5.2 - 180° rotation test. ....   | 65 |
| Figure 5.3 - The 2 icebergs used for testing in their initial positions after being processed using SPTMesh. ....             | 66 |
| Figure 5.4 - Iceberg 1 interpolation test over an interval of time. ....  | 66 |

|  |    |
|--|----|
| Figure 5.5 - Iceberg 2 interpolation test over an interval of time. ....   | 66 |
| Figure 5.6 - The evolution of the 2 icebergs, seen together, over an interval of time. ....  | 67 |
| Figure 5.7 - Ice 1 at instants 1000 and 2000. ....   | 68 |
| Figure 5.8 - Ice 2 at instants 1100, 2000, 3000, 4000 and 4500. ....   | 69 |
| Figure 5.9 - Ice 1 at instant 1500. ....   | 69 |
| Figure 5.10 - Ice 1 area during the PERIOD(1100 10000). ....   | 70 |
| Figure 5.11 - 2 Icebergs intersecting over an interval of time. ....   | 71 |
| Figure 5.12 - ST_Intersection result at instant 1000 in light blue, seen in QGIS. ....   | 71 |
| Figure 6.1 - Examples of cloned (left) and collinear (right) vertices. ....  | 75 |
| Figure 6.2 - 3 line segments configurations. ....  | 76 |
| Figure 6.3 - Cloned vertex shifted along the line segments. ....   | 76 |
| Figure 6.4 - Possible new positions for the cloned vertex shifted along the perpendicular line. ....   | 76 |
| Figure 6.5 - An invalid geometry. Source: smoothing_test_ice1_2_3_GEOS_Clones.m. ....  | 77 |
| Figure 6.6 - From left to right: source geometry, interpolated geometry at an instant and the target geometry. ....  | 78 |
| Figure 6.7 - From left to right: source geometry with a degenerated triangle, the interpolated geometry at 3 different instants and the target geometry. ....  | 78 |
| Figure 6.8 - From left to right: source geometry without the degenerated triangle, the interpolated geometry at 3 different instants and the target geometry. ....                                     | 78 |
| Figure 6.9 - From left to right: instants 1, 8 and 11. Coil interpolation test during an interval $I$ with a carefully chosen permutation of $A$ . Source: interpolation_test_coil_tri_order_2.m. .... | 79 |
| Figure 6.10 - From left to right: instants 1, 8 and 11. Coil interpolation test during an interval $I$ with a random permutation of $A$ . Source: interpolation_test_coil_tri_order.m. ....            | 79 |
| Figure 6.11 - A sequence of triangles with ids from 1, ..., 18. ....   | 79 |
| Figure 6.12 - Triangulated geometry of an iceberg. ....  | 80 |
| Figure 6.13 - Interpolation using the smoothing method. ....   | 81 |
| Figure 6.14 - Interpolation not using the smoothing method. ....   | 81 |
| Figure 6.15 - A source and target relatively simple smoothed meshes. ....  | 81 |



---

## List of Tables

|  |    |
|--|----|
| Table 1.1 - Dissertation structure and content. ....   | 4  |
| Table 2.1 - Main concepts used in this dissertation. ....  | 6  |
| Table 3.1 - The signature of our discrete data model for moving objects. ....                      | 23 |
| Table 3.2 - Notation. ....   | 23 |
| Table 3.3 - Type <i>mesh</i> components. ....  | 26 |
| Table 3.4 - The set of operations on MOVING types in our discrete model. ....                      | 29 |
| Table 3.5 - Predicate operations. ....   | 29 |
| Table 3.6 - Set operations. ....   | 30 |
| Table 3.7 - Numeric operations. ....   | 30 |
| Table 3.8 - Projection of MOVING types to Domain and Range. ....                                   | 30 |
| Table 3.9 - Restriction of MOVING types to the time and spatial domains. ....                      | 30 |
| Table 3.10 - Constructors. ....  | 31 |
| Table 4.1 - Implemented components. ....   | 34 |
| Table 4.2 - Technologies used to implement SPTMesh and MeshGIS. ....                               | 35 |
| Table 4.3 - Development environment setup. ....  | 35 |
| Table 4.4 - SPTMesh external dependencies. ....  | 41 |
| Table 4.5 - SPTMesh data structures to implement our discrete data model data types. ....          | 41 |
| Table 4.6 - Other relevant SPTMesh data structures. ....   | 42 |
| Table 4.7 - Notation. ....   | 43 |
| Table 4.8 - Predicate operations. ....   | 44 |
| Table 4.9 - Set operations. ....   | 44 |
| Table 4.10 - Numeric operations. ....  | 44 |
| Table 4.11 - Projection of MOVING types to Domain and Range. ....                                  | 44 |
| Table 4.12 - Interaction of MOVING types with values in Domain and Range. ....                     | 44 |
| Table 4.13 - Constructors. ....  | 45 |
| Table 4.14 - Other interesting operations. ....  | 46 |
| Table 4.15 - Triangulation, smoothing and interpolation methods implementation classes. ....       | 46 |
| Table 4.16 - The SPTMesh values used to help establish a notion of continuity for UNIT types. .... | 47 |
| Table 4.17 - Notation. ....  | 48 |
| Table 4.18 - STWKT formats for UNIT and MOVING types. ....   | 49 |
| Table 4.19 - Main classes and responsibilities. ....   | 51 |
| Table 4.20 - MeshGIS external dependencies. ....   | 53 |
| Table 4.21 - PostGIS data structures. ....   | 54 |

|   |    |
|---|----|
| Table 4.22 - MeshGIS data structures to represent moving objects. ....  | 55 |
| Table 4.23 - MeshGIS data structures to represent the interpolation components. ....  | 55 |
| Table 4.24 - MeshGIS and its binding SQL operations for SPTMesh MovingBool objects. ....  | 57 |
| Table 4.25 - MeshGIS and its binding SQL operations for SPTMesh MovingReal objects. ....  | 58 |
| Table 4.26 - MeshGIS and its binding SQL operations for SPTMesh MovingPoint objects. ....   | 58 |
| Table 4.27 - MeshGIS and its binding SQL operations for SPTMesh MovingMesh objects. ....  | 59 |
| Table 4.28 - PostGIS external dependencies. ....  | 60 |
| Table 4.29 - SPTMesh namespaces. ....   | 62 |
| Table 4.30 - SPTMesh project structure. ....  | 62 |
| Table 4.31 - MeshGIS project structure. ....  | 63 |
| Table 4.32 - Adding a new smoothing method to SPTMesh. ....   | 64 |
| Table 5.1 - Results of the select command in the icebergs table. ....   | 67 |
| Table 5.2 - Result of the ST_get_Size function after the icebergs' data was stored in the icebergs table. ....  | 68 |
| Table 5.3 - Period in which the records in the icebergs table are defined. ....   | 68 |
| Table 5.4 - Records in the icebergs table defined in a given period. ....   | 69 |
| Table 5.5 - Records in the icebergs table defined at a given instant. ....  | 69 |
| Table 5.6 - Area of the icebergs in a given instant. ....   | 70 |
| Table 5.7 - Evolution of the area of the icebergs in the table icebergs in a given period. ....   | 70 |
| Table 5.8 - ST_Intersect results. ....  | 71 |
| Table 5.9 - Using PostGIS to get the area of the intersection of 2 moving objects at instant 1000. ....   | 72 |
| Table 5.10 - ST_Intersection result when no intersection exists. ....   | 72 |
| Table 5.11 - ST_Present result at instant 2100. Ice 3 and 4 are not defined at that instant. ....   | 72 |
| Table 5.12 - ST_Get_Present_AtPeriod result at a given period. (x y 0) means that the object is not defined in the interval [x, y[. ....                                    | 72 |
| Table 6.1 - Results from a performance test performed in PostgreSQL using and not using the smoothing method. ....  | 82 |
| Table 6.2 - Centroid positions, in the 2D Cartesian plane, of 9 continuous unit mesh objects, obtained from real data, and their distances at the continuity instants. .... | 83 |
| Table 6.3 - The area differences between 9 continuous unit mesh objects. ....   | 83 |

---

## List of Acronyms

|              |  |               |  |
|--------------|--|---------------|--|
| <b>ADT</b>   | Abstract Data Type                       | <b>OGC</b>    | The Open Geospatial Organization             |
| <b>ADAS</b>  | Advanced Driver Assistance Systems       | <b>OODBMS</b> | Object-Oriented Database Management System   |
| <b>DBMS</b>  | Database Management System               | <b>ORDBMS</b> | Object Relational Database Management System |
| <b>DDL</b>   | Data Definition Language                 | <b>OSGeo</b>  | Open Source Geospatial Foundation            |
| <b>DML</b>   | Data Manipulation Language               | <b>STIS</b>   | Spatiotemporal Information System            |
| <b>FTL</b>   | Future Temporal Logic query language     | <b>STDBMS</b> | Spatiotemporal Database Management System    |
| <b>GPS</b>   | Global Positioning System                | <b>TDs</b>    | Trajectory Databases                         |
| <b>GMODs</b> | Generic Moving Objects Databases         | <b>TDWs</b>   | Trajectory Data Warehouses                   |
| <b>IDE</b>   | Integrated Development Environment       | <b>SQL</b>    | Structured Query Language                    |
| <b>IoT</b>   | Internet of Things                       | <b>SVD</b>    | Singular Value Decomposition                 |
| <b>LBSs</b>  | location-based services                  |               |  |
| <b>MOs</b>   | Moving Objects                           |               |  |
| <b>MOST</b>  | Moving Objects Spatiotemporal data model |               |  |



---

## Introduction

There is an emergence of a growing number of applications and services based on spatiotemporal data in the most diverse areas of knowledge and human activity. The internet of things (IoT), the emergence of technologies that make it possible to collect information about the evolution of real world phenomena and the widespread use of global positioning system (GPS) enabled devices, such as smartphones and navigation systems, suggest that the volume and value of these data will increase significantly in the future. In this context, it is necessary to develop tools capable of extracting knowledge from these data and efficient ways to manage them: represent, manipulate, analyze and store. But this data can be complex, its management is not trivial and there is not yet a complete system capable of performing this task.

Database management systems (DBMSs) are a natural environment to manage data in an efficient way. However, existing DBMSs are not prepared to manage spatiotemporal data. One solution is to use DBMSs that provide extensible architectures that allow the implementation of extensions, i.e., user-defined data models, to manage spatiotemporal data.

The management of spatiotemporal data, in DBMSs, has been studied from different perspectives, using different techniques and architectures. Several data models have been proposed and there is already a considerable amount of work done and published on this subject. However, this topic is still an open field for research.

Spatiotemporal databases have applications in several fields:

- Advanced driver assistance systems (ADAS) and intelligent transportation systems (Popa & Zeitouni, 2012).
- Environmental information systems (Wahid, Kamruzzaman, & Shariff, 2006).
- Location-based services (LBS) (Nikos Pelekis, Frentzos, Giatrakos, & Theodoridis, 2008; Nikos Pelekis & Theodoridis, 2006).
- Vehicle traffic analysis (Nikos Pelekis, Frentzos, Giatrakos, & Theodoridis, 2010).
- Air traffic control (Hurter, Andrienko, Andrienko, G, & Sakr, 2013).

In this chapter we present: the context, the motivation, the goals, the structure and content of this dissertation.

## 1.1 Context, Motivation and Goals

Working with spatiotemporal data, in particular data that represents the continuous evolution (i.e., the changes in position, shape and extent) of some entity (or object) over time is challenging due to several reasons (Amaral, 2015):

- The need to acquire 2D geometric representations of these entities from real data, e.g., from satellite images.
- The impossibility to store unlimited amounts of data. As a consequence, we have to take observations at specific instants in time and represent these continuous changes from that set of discrete data. This requires the existence of some interpolation function that should have the following properties:
  - Causes minimal deformation and preserves the physical characteristics of the entity being represented. That is, the function should provide a realistic representation of the entity's changes in position, shape and extent over time. And this function is not necessarily a simple function, e.g., find a function to represent a realistic evolution of an iceberg between 2 known observations (see Figure 1.1).
  - Generates only valid geometry topologies, i.e., generates geometries with no self-intersections.

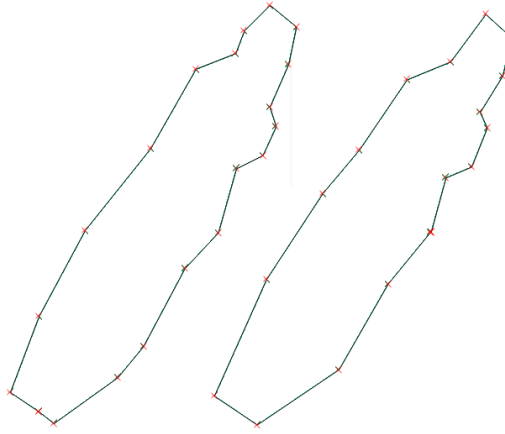


Figure 1.1 - 2 observations of the evolution of an iceberg over time.

Previous dissertations, (Paulo, 2012), (Mesquita, 2013) and (Amaral, 2015), studied and proposed methods to deal with these issues and this dissertation uses them as the foundation to establish a framework for moving objects, in particular objects whose position, shape and extent change continuously over time. These methods enable us to obtain spatiotemporal data from real data and a realistic evolution of that data over time. In this dissertation we want to be able to manipulate, analyse and store this data and to achieve that:

- We propose a discrete data model for moving objects based on the methods implemented on these dissertations.

- We implement the proposed data model in a framework for moving objects that is independent from any client using it.
- We implement a spatiotemporal extension for PostgreSQL that uses our framework for moving objects.

The main goals of this dissertation are the following:

- Propose and establish a discrete data model for moving objects that uses the concept of mesh to represent objects whose position, shape and extent change continuously over time. Such data model is independent from any specific client or application using it, and that includes DBMSs.
- Implement the proposed discrete data model using the methods implemented in (Amaral, 2015) as part of our framework for moving objects and evaluate its results using real data obtained by the methods implemented in (Paulo, 2012) and (Mesquita, 2013).
- Build a spatiotemporal database extension as a proof of concept that our framework works with real applications and can be used to implement spatiotemporal extensions.
- Establish a framework for moving objects that can be used as a reference for future work and investigation in this area.

## 1.2 Outline of this Dissertation

This dissertation is structured in 7 chapters, see Table 1.1.

| Chapter | Main Objectives and Content  |
|---------|--|
| 2       | <p>In this chapter we review related work found in the literature, in particular about:</p> <ul style="list-style-type: none"> <li>• Spatiotemporal databases: applications, architectures, data models and languages.</li> <li>• Spatiotemporal query languages.</li> <li>• Spatiotemporal object relational database management systems (ORDBMSs).</li> </ul> <p>And we briefly discuss the problems of representing the evolution of objects whose position, shape and extent change continuously over time and morphing and shape interpolation methods.</p> |
| 3       | <p>In this chapter we present and propose a discrete data model for moving objects that uses the concept of mesh to represent objects whose position, shape and extent change continuously over time and we define its data types and operations.</p>  |
| 4       | <p>In this chapter we discuss implementation details:</p> <ul style="list-style-type: none"> <li>• Our main goals and implementation specific needs.</li> <li>• The technologies and architectures that we considered using, their advantages and disadvantages.</li> <li>• The technologies we used in our implementation.</li> </ul>   |

---

|   |   |
|---|---|
|   | <ul style="list-style-type: none"> <li>• The implementation details of the components that were implemented: its dependencies, data structures, operations, architecture, usage and installation and implementation issues.</li> </ul> <p>We also discuss the structure of the implemented components source code and the extension of our framework for moving objects with new triangulation and smoothing methods.</p> |
| 5 | In this chapter we present and discuss the tests that were performed to test and validate the components and the methods that were implemented.   |
| 6 | In this chapter we present and discuss the major problems that were found, in particular, when working with real data. We present observations, opinions, possible solutions and explanations based on experimentation. And we discuss the structure of some of these problems to provide future work on these issues with some initial background on such a structure.   |
| 7 | In this chapter we present our main conclusions and discuss future work.  |

---

Table 1.1 - Dissertation structure and content.



## Representation and Management of Spatiotemporal Data

The management of spatiotemporal data, in DBMSs, has been studied from different perspectives, using different techniques and architectures. Several solutions have been proposed and there is already a considerable amount of work done and published on this subject. However, this topic is still an open field for research.

In this chapter we start by presenting some of the work done on the field of spatiotemporal databases: their applications and architecture, data models, operations and query languages. It is not, however, a general overview on the subject. We focus our presentation on work related to:

- The representation of moving objects (MOs) that evolve continuously over time in unconstrained space, using abstract data types (ADTs).
- The management of the entire history of the evolution of MOs, i.e., its position or shape and extent changes over time.
- Data models that are not application-specific and that are capable of representing both: moving points and moving regions.
- Data models that extend object-relational database management systems (ORDBMSs) with spatiotemporal functionality, i.e., data models using an extensible architecture.
- Query languages for spatiotemporal databases.

Then we discuss some of the problems of working with moving regions, morphing and triangulation methods.

Other subjects, e.g., MOs moving in networks, constraint databases or MOs with different transportation modes, will be briefly discussed, when found relevant, or omitted.

### 2.1 Concepts

Throughout this dissertation we use some concepts that we define in this subsection. See Table 2.1.

| Concept             | Definition  |
|---------------------|---|
| Spatiotemporal Data | Data that has spatial and temporal components or aspects. It is usually represented in 3D, i.e., time + space (2D), can be highly complex and can be associated with metadata. Represents the discrete or continuous evolution, |

|                         |  |
|-------------------------|--|
|                         | where evolution refers to the changes in position or shape and extent, of some entity or object over time. Some examples of spatiotemporal data include: a hurricane trajectory over time, a ship trajectory during a storm, the persons' movements during the day, the evolution of cells and biological tissues over time.   |
| Moving Point            | A moving entity (or moving object) for which only the position in space is relevant, e.g., the position of animals, people, aircrafts or ships.  |
| Moving Region           | A moving entity (or moving object) for which the position in space and the shape and extent, e.g., growing and shrinking, are relevant, e.g., hurricanes, icebergs and forest fires.   |
| Moving Object (MO)      | Used in the literature with different meanings. It can be used to refer to a moving point, a moving region or both. It is also used to represent entities or phenomena changing in discrete steps or continuously over time. In this dissertation, a MO is an entity (or object), whose position or shape and extent change continuously over time. When we want to refer specifically to moving points or to moving regions we do so explicitly. The same is true when we want to refer to MOs that change in discrete steps or continuously over time. |
| Spatiotemporal Database | A database system or a database extension that provides services to represent, manipulate, analyse and store spatiotemporal data in an efficient and convenient way, e.g., efficient data storage and representation, query optimization and indexing services.  |

Table 2.1 - Main concepts used in this dissertation.

## 2.2 Spatiotemporal Databases

IoT and the widespread use of GPS-enabled devices, such as smartphones and navigation systems, are an opportunity for the development of new services and applications, e.g., location-based services (LBSs), advanced trip planning and recommendation systems, fleet management and logistics and to study real world phenomena, e.g., people and animal movements and traffic evolution.

There is also an increasing range of applications that deal with specific real world phenomena, whose position, shape and extent change discretely or continuously over time, e.g., pollution areas, hurricanes and land usage. These applications due to their impact are important for several areas of human knowledge.

These applications and services need an efficient and convenient storage, representation, analysis and manipulation (providing suitable operations), given their potential to produce large amounts of spatiotemporal data that can have complex data structures and to ease the extraction of knowledge from spatiotemporal data and all the information it carries.

DBMSs were built to manage data in an efficient and convenient way, however, existing DBMSs cannot deal with spatiotemporal data off-the-shelf and its management is a complex task.

People studying the management and representation of spatiotemporal data in DBMSs realized that temporal and spatial databases were connected and that it made sense to create a database system that integrated the temporal and spatial components together. So, a possible solution for the management of spatiotemporal data is to develop a spatiotemporal database management system (STDBMS) and such a system can provide support for the services and applications that we mentioned earlier.

There has been a lot of research on spatiotemporal databases. However, to the best of our knowledge, spatiotemporal database systems exist only at the prototype level in Secondo (R. H. Güting, Behr, & Düntgen, 2010) and Hermes (Nikos Pelekis, Theodoridis, Vosinakis, & Panayiotopoulos, 2006). The design and implementation of a complete STDBMS is a complex task that requires among other things: efficient algorithms for query operations on MOs, indexing techniques, query optimization, an efficient representation and storage and a realistic representation of the evolution of moving regions over time.

### **2.2.1 Applications**

Spatiotemporal databases have applications in several fields:

- Social networks (K.-S. Kim, Ogawa, Nakamura, & Kojima, 2014).
- Advanced driver assistance systems (ADAS) and intelligent transportation systems (Popa & Zeitouni, 2012).
- Environmental information systems (Wahid et al., 2006).
- Location-based services (LBS) (Nikos Pelekis et al., 2008; Nikos Pelekis & Theodoridis, 2006).
- Vehicle traffic analysis (Nikos Pelekis et al., 2010).
- Air traffic control (Hurter et al., 2013).
- Maritime surveillance and security (Etienne, Devogele, & Bouju, 2010).
- Trip planning and recommendations systems (Booth, Sistla, Wolfson, & Cruz, 2009; Zheng, Zhang, Ma, Xie, & Ma, 2011).

### **2.2.2 Architecture**

In (Breunig et al., 2003) the authors present the main system architectures to build a STDBMS: the layered architecture, the monolithic architecture and the extensible architecture. According to the authors these architectures can be described as follows.

The layered architecture implements a layer on top of an off-the-shelf DBMS, to provide spatiotemporal functionality. There is a clear separation of responsibilities between the two layers, i.e., the DBMS layer and the layer providing the spatiotemporal functionality. Transaction management, query processing and optimization and indexing services for the standard data types, handled by the DBMS, are provided by the DBMS level. The new layer has to provide these services for the non-standard data types. Global query

processing and optimization of combined queries is hard to implement and combined index processing cannot be used.

The main advantage of this architecture is to enable a fast development of new services (Matos, Moreira, & Carvalho, 2012).

In the monolithic architecture, standard and non-standard data types and operations are integrated into the DBMS kernel. Because of this tight integration most of the problems found in the layered architecture are overcome. However, implementing such an integrated system, for spatiotemporal data management, has proved to be a hard task. Monolithic systems can be optimized for specific application domains.

The extensible architecture gathers the best features of the previous ones, to provide a DBMS that allows user-defined extensions (data types and operations) to be plugged into the database system. The new data types and operations are integrated into the DBMS, but outside its kernel. Indexing services, query optimization and transaction management can be provided at the DBMS level for the new data types and they can be manipulated using the structured query language (SQL). Developing new extensions is easier than developing new services inside the DBMS kernel.

The use of this architecture, in general, implies the use of a DBMS that provides extensibility capabilities and several DBMSs already support some kind of extensibility technology, e.g., Informix datablades, Oracle cartridges, IBM DB2 extenders. Secondo (R. H. Güting et al., 2010) is also an example of a system using the extensible architecture.

### **2.2.3 Data Models and Operations**

Several spatiotemporal data models have been proposed in the literature (Nikos Pelekis, Theodoulidis, Kopanakis, & Theodoridis, 2004). Some use existing spatial and temporal data models to develop a spatiotemporal data model. Others propose their own modelling approach. Not all have been formally defined or properly evaluated. Some have not been implemented, are incomplete or in an experimental stage. And, in general, they deal with the needs of specific applications and services.

Our discussion focuses on moving objects data models, since we are interested in the representation of MOs. Moving objects data models can capture the continuous evolution of moving entities over time, i.e., the changes in their position, shape and extent.

#### **Moving Objects Data Models**

MOs have been studied from different perspectives (R. Güting & Schneider, 2005):

- The location management perspective: i.e., manage the position of MOs, ask questions about the current and expected near future positions and the relationships that might develop between objects. No history of movement is kept.
- The spatiotemporal database perspective: i.e., describe the current state of MOs and the whole history of their evolution (the position in the near future might also be considered), understand how

things changed, analyze when certain relationships occurred and retrieve the state at a particular instant in the past. Changes in shape and extent are also considered. Our discussion will be centered in the analysis of data models using this perspective.

In the following we will briefly discuss data models using the location management perspective, in particular the MOST data model (Prasad Sistla, Wolfson, Chamberlain, & Dao, 1997), and then we will discuss data models using the spatiotemporal database perspective in more depth.

### **The Location Management Perspective**

In (Prasad Sistla et al., 1997) the authors proposed the moving objects spatiotemporal (MOST) data model.

According to the authors in (R. H. Güting, De Almeida, & Ding, 2006), MOST allows to model and query the current and near future movement of MOs, in a database. Positions are not stored directly, avoiding a high volume of updates. Instead, the database holds a motion vector and updates are executed, only, when the position predicted by the motion vector deviates from the real position by more than a given threshold. The model introduces the concept of dynamic attributes, i.e., attributes that change implicitly over time, leading to a notion of continuous queries.

In (Prasad Sistla et al., 1997) the authors also proposed the future temporal logic (FTL) query language that makes it possible to specify temporal relationships between objects and to formulate queries about near future movements. The MOST data model deals, only, with moving points.

### **The Spatiotemporal Database Perspective**

According to the authors in (Matos et al., 2012):

- The most well-known paradigms used to develop spatiotemporal data models and query languages are based on constraint databases (Grumbach, Rigaux, & Segoufin, 2001) and abstract data types (ADTs) (Cotelo Lema, Forlizzi, Güting, Nardelli, & Schneider, 2003; Forlizzi, Güting, Nardelli, & Schneider, 2000; R. H. Güting et al., 2000).
- The constraint databases data model approach main disadvantage is that it can hardly be integrated into ORDBMSs. This model has, however, very interesting properties, e.g., allows the representation of spatiotemporal data of arbitrary dimension and offers uniformity in its representation, i.e., objects are represented using linear constraints.
- The ADTs based data models can be smoothly integrated into ORDBMSs (Matos et al., 2012) and the most well-known approach for the representation of spatiotemporal data in databases, proposed in (R. H. Güting et al., 2000), uses ADTs.

In (R. H. Güting et al., 2000) the authors propose an abstract data model, a system of data types and appropriate operations forming an algebra that can be embedded in a query language, that sets a solid foundation for implementing a spatiotemporal DBMS extension. According to the authors in (Matos et al.,

2012), an interesting property of this model is that complex data types are constructed from simpler ones, which is particularly appropriate for implementation on extensible database architectures.

The abstract data model type system defines the base types ( $\text{BASE} = \{\text{int}, \text{real}, \text{string}, \text{bool}\}$ ), the spatial types ( $\text{SPATIAL} = \{\text{point}, \text{points}, \text{line}, \text{region}\}$ ), the time type ( $\text{TIME} = \{\text{instant}\}$ ), temporal types that are derived from  $\text{BASE} \cup \text{SPATIAL}$  using a moving and an intime type constructors and range types whose values are finite sets of pairwise disjoint intervals over the domain  $\text{BASE} \cup \text{TIME}$ . The most important types are moving point and moving region.

It also proposes a set of operations, on temporal and non-temporal types, to support querying and a generalized definition of continuity valid for all the proposed temporal data types, where discontinuity represents the case of values changing in discrete steps.

In (Forlizzi et al., 2000) the authors propose a discrete data model and data structures, that can be mapped into concrete physical data structures in a DBMS environment, that implements the abstract data model proposed in (R. H. Güting et al., 2000). This paper introduces the concept of the *sliced representation* (see Figure 2.1 and Figure 2.2), to represent the temporal types, i.e., the temporal evolution of a value is decomposed into fragments (units) called slices. A slice records the evolution of a value  $v$  of some type  $\alpha$  in a given time interval  $I$ , using a simple function defined in  $I$ , and maintains type-specific constraints during that evolution. A unit type holds a single slice and a moving type is a set of unit types, whose time intervals are mutually disjoint. This allows the representation of the evolution of continuous phenomena over time using data taken at specific instants in time, i.e., using discrete data.

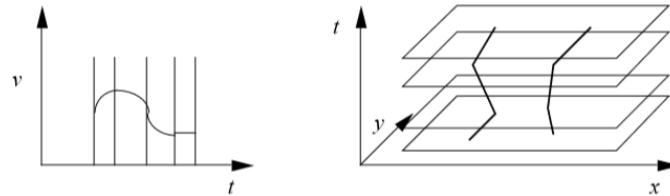


Figure 2.1 - Sliced representation of: a moving real (left) and 2 moving points (right). Source: (Forlizzi et al., 2000).

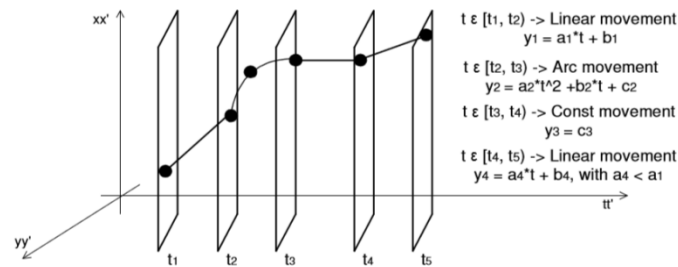


Figure 2.2 - A Moving point with various types of movement. Source: (Nikos Pelekis & Theodoridis, 2007).

In (Tøssebro & Nygård, 2011) the authors propose an extension to the *sliced representation*, proposed in (Forlizzi et al., 2000), capable of representing explicit topology. The authors present an analysis to determine

which topological relationships need to be stored explicitly and which can be computed from the geometry. They also take into consideration that storing certain topological relationships, to make sure that the interpolation of neighboring geographical objects matches over time, might be of interest. The paper describes how to represent topology for continuously moving or changing objects, how to deal with objects that are connected topologically but not necessarily updated at the same time and how to represent changes in topology over time.

## **Operations**

In (R. H. Güting et al., 2000) the authors present a carefully designed set of operations on non-temporal and temporal types defined at an abstract level. According to the authors the design of these operations aims to achieve three main goals: design operations that are as generic as possible, achieve consistency between operations on non-temporal and temporal types and capture the interesting phenomena. The non-temporal operations are transformed, using a process of lifting, into temporal operations. The classes of operations on temporal types include: projection to domain and range, interaction with points and point sets in domain and range, the when operation and rate of change operations. The authors also present operations on sets of objects and provide a complete, precise definition of the signature of all the proposed operations. According to the authors, the idea of lifting is based on the definition of an algebra over non-temporal types that is transformed into operations over temporal types and the use of this concept makes it possible to obtain consistency between temporal and non-temporal operations.

In (Cotelo Lema et al., 2003) the authors present a comprehensive and systematic study of algorithms for implementing a subset of the operations proposed in (R. H. Güting et al., 2000), using the data structures presented in (Forlizzi et al., 2000). Some of these algorithms are complex, e.g., set operations on moving regions and computations of distance functions involving moving regions.

### **2.2.4 Spatiotemporal Query Languages**

Query languages for spatiotemporal databases should provide an efficient, simple and natural way to ask any kind of questions about the movement and evolution of spatiotemporal objects over time. Of special interest are query languages able to express such queries about MOs. These query languages should allow querying the past and the present, and possibly the near future, evolution of MOs. According to the authors in (Pozzani & Combi, 2012), a complete query language for spatiotemporal databases does not yet exist and although several spatiotemporal query languages have been proposed, there have not been efforts for standardizing a spatiotemporal model and query language.

In (M. Erwig & Schneider, 1999; Martin Erwig & Schneider, 2002) the authors propose a spatiotemporal query language called STQL, as an extension of the SQL query language, that allows to formulate spatiotemporal queries about the development of topological relationships over time, using spatiotemporal predicates. Complex predicates can be built from a set of elementary ones. STQL provides a framework based on the notion of spatiotemporal predicates.

In (Xinmin Chen & Zaniolo, 2000) the authors present the SQL<sup>ST</sup> query language. SQL<sup>ST</sup> main objective is to minimize the extensions required in SQL, or other relational languages, to support spatiotemporal queries. According to the author in (Noh, 2004), SQL<sup>ST</sup> cannot deal with the specific features of spatiotemporal data in a natural and convenient way, because of the restrictions of the standard SQL query language to deal with spatiotemporal queries.

The STQL query language, proposed and presented in (D. H. Kim, Ryu, & Kim, 2000; D. Kim, Ryu, & Park, 2002), is based on the SQL3, TSQL2 and the Spatial SQL specifications. It consists on a data definition language (DDL) and a data manipulation language (DML). In (D. Kim et al., 2002) the authors propose a spatiotemporal query processing (STQP) system to process queries written with STQL. The STQP system uses a composite architecture and it is composed by a spatiotemporal syntax analyser, a spatiotemporal semantic analyser, a spatiotemporal code generator and a spatiotemporal interpreter.

In (R. H. Gutting et al., 2003) the authors present the design of the core of a spatiotemporal extension to SQL 92, called STSQL, that provides built-in data management support for spatiotemporal data and makes it possible for legacy database applications, using a conventional SQL 92-based DBMS, to be migrated to a STDBMS without affecting the legacy applications.

In (Viqueira & Lorentzos, 2007) the authors propose a SQL extension for the management of spatiotemporal data. The extension considers discrete changes in space and time. The syntax and semantics are fully consistent with the SQL 2003 standard and it is based on data types, defined in terms of time and spatial quanta and SQL constructs.

In (Pozzani & Combi, 2012) the authors present the spatiotemporal query language ST4SQL, to deal with temporal and spatial dimensions qualified with granularities. ST4SQL is a SQL-based query language that extends the SQL syntax and the T4SQL temporal query language. ST4SQL queries and constructs can be translated into equivalent standard SQL queries and constructs. ST4SQL introduces four temporal and spatial semantics that add a specific meaning to queries and allow the user to specify how the system has to manage temporal and spatial dimensions for evaluating queries.

## 2.3 Spatiotemporal ORDBMSs

Current ORDBMSs, e.g., Oracle, IBM DB2, Informix, PostgreSQL, SQL Server, MySQL, are built using extensible architectures that provide some kind of extensibility mechanism for user-defined data types and operations, index structures and query optimization. These extendible architectures can be used to integrate new data models, into a DBMS, and its data types can be manipulated using SQL, e.g., spatial extensions. These ORDBMSs are being used to develop spatiotemporal extensions and hence the name spatiotemporal ORDBMSs.

Several spatiotemporal extensions, built on top of ORDBMSs, have been proposed in the literature, covering different fields of application. In general, these extensions focus on specific applications and its implementation is not trivial. According to the authors in (Matos et al., 2012), these facts seem to indicate



that the management of spatiotemporal data, in standard DBMSs, is a relevant issue and the existence of general purpose spatiotemporal extensions would be useful in many domains of knowledge.

In this section we present, to the best of our knowledge, the most relevant ORDBMSs spatiotemporal extensions, using the ADTs approach, proposed in the literature.

Secondo (de Almeida et al., 2004; R. H. Güting et al., 2010) is an open-source, fully extensible, database system, built from scratch, that provides a generic environment for building database systems prototypes. It is, also, a research prototype, supporting spatial and spatiotemporal data management. Offers a collection of data structures and operations for representing and querying MOs and MOs can be visualized and animated. Secondo does not conform to the open geospatial consortium (OGC) standards and it does not follow a predefined data model. It can be extended with new data models and its core data model can be modified. According to the authors in (R. H. Güting et al., 2005), Secondo is, or was, used to study moving objects databases (MODs), network models, fuzzy spatial data types and optimization techniques.

Several data models have been implemented partially or completely as part of or using Secondo (Behr, Teixeira de Almeida, & Güting, 2006; Forlizzi et al., 2000; R. H. Güting et al., 2000; Hartmut, Zhiming, & Almeida, 2006; Xu & Güting, 2012). Secondo is also the base for other projects: Parallel Secondo (R. Güting & Lu, 2013), i.e., Secondo enhanced with parallel processing technologies for large scale processing and analysis of MOs data in a cluster of computers, and Distributed Secondo (Nidzwetzki & Güting, 2016), an extensible highly available and scalable database management system.

Hermes (Nikos Pelekis, Frentzos, Giatrakos, & Theodoridis, 2015; Nikos Pelekis & Theodoridis, 2007) is a framework designed as a system extension that provides spatiotemporal functionality to OpenGIS-compatible state-of-the-art ORDBMSs. It is, also, a research prototype for efficient location-based data management. Hermes defines a set of moving type objects and operations (Nikos Pelekis et al., 2006), using the temporal types defined in the TAU temporal literal library (TAU-TLL) (N Pelekis, 2002) and the spatial data types defined in the underlying ORDBMS OGC-compliant spatial extension. Hermes proposes a data type model and a complete set of state-of-the-art query processing algorithms for trajectory databases (TDs). Takes advantage of the extensibility mechanisms provided by ORDBMSs that offer an OGC-compliant spatial extension and it was implemented as an extension on two ORDBMSs: Oracle, using Oracle Spatial, and PostgreSQL. Supports real time dynamic applications, e.g., location-based services (LBS), and it has been successfully used in four different domains (Nikos Pelekis et al., 2015): trajectory data warehouses (TDWs), moving objects data mining query languages, semantic enrichment of movement patterns and privacy-aware trajectory tracking query engines.

In (Matos et al., 2012) the authors propose a spatiotemporal data model that was implemented and tested on Oracle 11g DBMS using Oracle 11g Spatial. This data model introduces modifications to the internal structure of the representation of moving regions, proposed in (Forlizzi et al., 2000; R. H. Güting et al., 2000), reducing the storage requirements and the size of the temporary data structures used in the evaluation of spatiotemporal operations. The authors also present algorithms to implement a subset of spatiotemporal operations, namely, projections and predicates, relying on the spatial features of the underlying DBMS. A

moving region is defined as an ordered collection of units, composed by a time interval and a set of moving points identifiers defining its geometry. Moving points may be shared by several moving regions, making it possible to update a moving point while the corresponding moving region(s) remain(s) unchanged. The costs of updates and the size of the data structures are reduced and the performance of the operations dealing with moving regions is improved.

The spatiotemporal object cartridge (STOC) data model, presented in (Zhao, Jin, Zhang, Wang, & Lin, 2011), is an extension developed on Oracle, based on Oracle Spatial, that provides support for spatiotemporal data management. The data model provides temporal and spatiotemporal data types and operations and supports both, continuous and discrete changes of spatial data over time and the following query types: temporal range, spatial range, spatiotemporal range, spatiotemporal distance, spatiotemporal topology and spatiotemporal aggregate queries.

The Oracle-based spatiotemporal module (OSTM) (P Jin & Sun, 2008) is an extension implemented as a data cartridge on top of Oracle providing spatiotemporal data management. It is based on the STORM (Peiquan Jin, Yue, & Gong, 2005) data model (also proposed by the authors). That is, OSTM can be thought of as an implementation of the STORM data model on Oracle. In the STORM data model the manipulation of spatiotemporal data is accomplished by extending the relational algebra with spatial, temporal and spatiotemporal operations. Complex regions, point sets, and polylines are not defined. STORM defines five basic extended relational algebraic operations (spatiotemporal union, spatiotemporal difference, spatiotemporal product, spatiotemporal selection and spatiotemporal projection) that form a complete set of relational algebra (therefore, other operations can be implemented using these five basic operations). STORM supports five types of spatiotemporal changes, described as: continuous and discrete spatial processes, continuous and discrete thematic processes and discrete life.

The component based moving region (CMR) data model (McKenney, Viswanadham, & Littman, 2014) is a data model to represent moving regions. According to the authors, the data model aligns well with data collection techniques, can be implemented easily and allows complex movement patterns to be easily described. The semantic interpretation of a region is separated from its physical representation, e.g., users are able to create a region containing what is usually considered to be an invalid structure. The model is designed to take advantage of simple algorithms for implementing operations on MOs, focusing on using well-known 2D algorithms instead of more complex 3D algorithms. A moving region is defined as a set of interval regions and an interval region describes the motion of a region over a specified time interval. A structural region contains components that, without interpretation, define an invalid region, i.e., they must be interpreted using an extraction function to extract the valid region. A structural region is defined as four sets: a set of faces, a set of holes, a set of lines, and a set of points, as a separate structural representation and interpretation and lays the foundation for the data model. The model assumes a single movement function for all moving regions in a particular application. The authors plan to extend the model to include moving points and lines.

The *balloon* data model (Praing & Schneider, 2007) uses the metaphor of a balloon to model the evolution of MOs over time: the string and the body of the balloon object represent the past and the future evolutions, respectively. The connection point between the string and the body of the balloon represents the present state. The authors discuss problems such as, appearing, disappearing, splitting and merging of object components and they provide a precise specification of the properties of the evolution of MOs, making clear how MOs can evolve in space and time, i.e., they define valid and invalid evolutions. Examples of invalid evolutions are instantaneous shrinking and instantaneous appearance. The model can represent the set of potential (predicted) future positions of a MO (that can be a simple or a complex spatial object). For that the authors introduce the concept of *confidence distribution*. The *confidence distribution* concept allows associating each potential future position of a MO with a degree of confidence. The model supports past and future queries and queries that start in the past and extend into the future. It's a general purpose model that does not provide a prediction model for the future evolution of MOs. Although an implementation is not provided, the authors plan to implement the *balloon* model as part of their spatiotemporal algebra (STAL) software package.

## 2.4 Representation of Moving Regions

The *sliced representation*, (Forlizzi et al., 2000), is used to represent the continuous evolution of MOs. However, the representation of the evolution of moving regions, in particular, is complex. An example follows.

Given a moving region evolving continuously over time, e.g., an iceberg, when using the *sliced representation* we proceed as follows. We take observations from its evolution at specific instants in time. In Figure 2.3 we can see, on top, the observations that were taken, below, the instant in time where each observation was taken at and the slices represented in green.

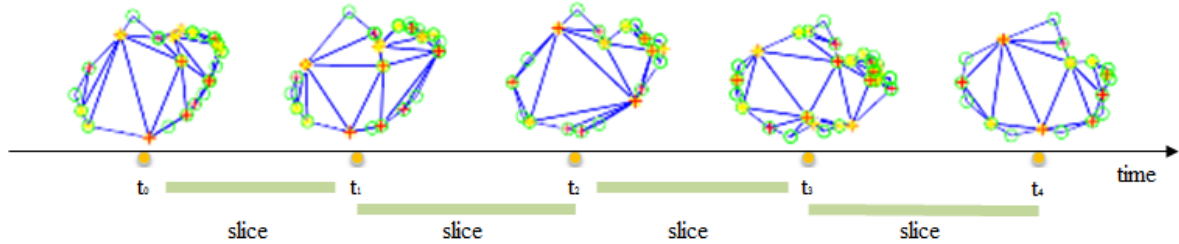


Figure 2.3 - Observations of an iceberg evolving continuously over time.

Then, we need a function  $\phi$  to give us the evolution of the region during each slice.  $\phi$  should have certain properties of interest, in particular:

- It should preserve the physical characteristics of the object. We are interested in a realistic and as accurate as possible representation of the region's evolution. Figure 2.4 shows examples of possible interpolation results, purple dots, that represent the region's evolution during a slice. The physical characteristics of the object are not preserved generating undesirable results and an unrealistic representation of the regions' evolution over time.

- It should generate no invalid geometries, i.e., geometries with no self-intersections.

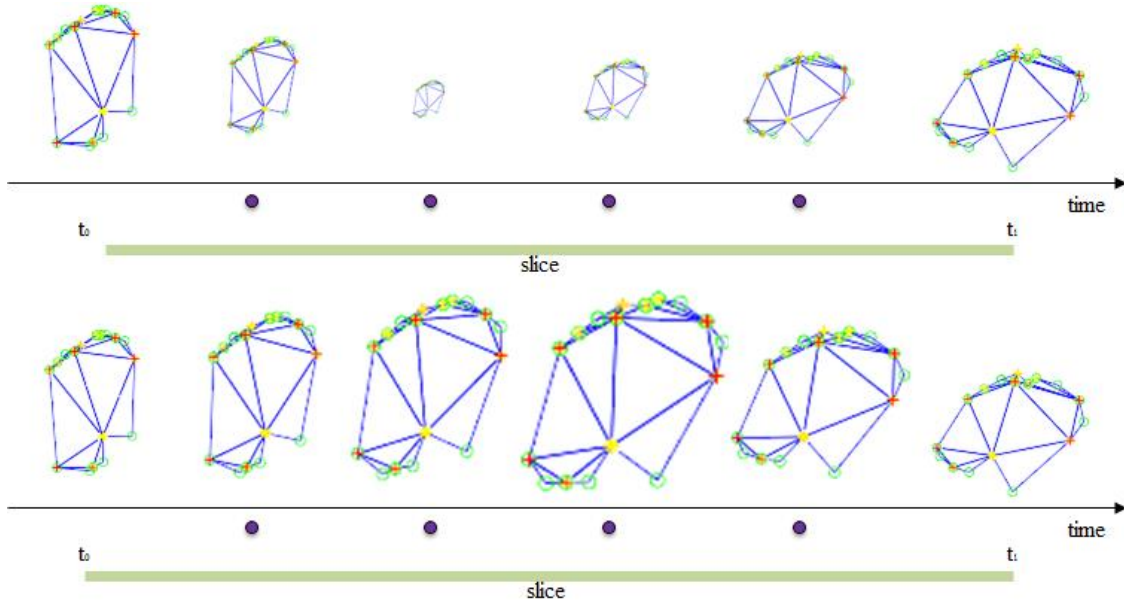


Figure 2.4 - Examples of possible interpolation results that represent the region's evolution during a slice.

The use of a simple function will not produce the desired results for most cases, i.e., a simple function is not adequate to obtain a realistic representation of the evolution of moving regions. The solution is to use more advanced techniques, e.g., morphing and shape interpolation methods. Figure 2.5 shows the interpolation between two known states (observations) of a region at time instants  $t_1$  and  $t_2$  using two different morphing techniques (Baxter, Barla, & Anjo, 2008). The technique shown on top might yield undesirable results, depending on the phenomena being studied.

Finding a suitable  $\phi$  for all possible cases is a complex task with some well-known problems, e.g., the growing and shrinking effect shown in Figure 2.4 and invalid geometries. These problems have been studied and there are proposals in the literature for function  $\phi$  but they don't work for all cases.

We are not going to discuss these advanced techniques and the problems that arise when trying to find  $\phi$  in this dissertation, but we leave some interesting references here for the interested reader:

- Works on finding  $\phi$ : (Heinz & Güting, 2016; Mckenney & Webb, 2010; Mckennney & Frye, 2015; Tøssebro & Güting, 2001)
- Morphing techniques using polygons compatible triangulations and rigidity-preserving interpolation methods: (Alexa, Cohen-Or, & Levin, 2000; Baxter et al., 2008; Gong, 2011; Craig Gotsman & Surazhsky, 2001; Haesevoets & Kuijpers, 2004).
- Dissertations on the study and implementation of methods to get spatiotemporal data from images of real world phenomena and defining  $\phi$  using compatible triangulation methods and rigidity-preserving interpolation methods: (Paulo, 2012), (Mesquita, 2013) and (Amaral, 2015).

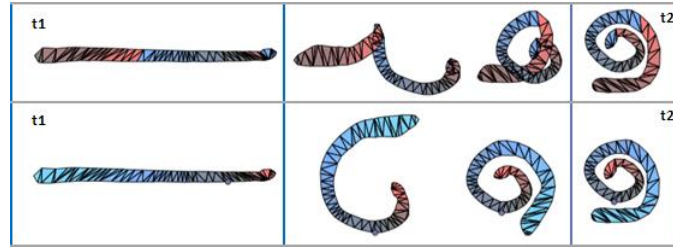


Figure 2.5 - Two different morphing techniques. Source: image adapted form (Baxter et al., 2008).

## 2.5 Summary

Spatiotemporal data is becoming more important for an increasing number of services and applications. With the rise of IoT and the widespread use of GPS-enabled devices its importance and value is expected to increase. We need tools to extract knowledge from these data and take full advantage of the information that they carry. For this to be possible, we need a system that can manage spatiotemporal data in an efficient and convenient way.

Spatiotemporal data can be highly complex and its management is not trivial. Several spatiotemporal data models and query languages have been proposed in the literature and there is already a considerable amount of work done and published on this subject. However:

- A complete system to manage spatiotemporal data does not yet exist. Only a few prototypes exist: Secondo and Hermes.
- There is no standard for a spatiotemporal data model and query language. Although no standard for the representation of moving objects exist, the open geospatial consortium (OGC) moving features standard (OGC Moving Features, 2016) specifies standard encoding representations of the movement of geographic features for information exchange. This standard applies to features that move as rigid bodies. It does not apply to all types of moving features, e.g., deforming features, therefore is not suitable for moving regions that change their shape over time.
- A lot of work on moving points has already been made. However, few data models exist to manage moving regions evolving continuously over time. This seems to be related to two main reasons (McKenney et al., 2014), generating moving region data from sensors is challenging and the algorithms proposed to implement operations on moving regions are complex.
- Several other challenges remain, such as, the development of indexing and query optimization techniques, the implementation of spatiotemporal operations on MOs and the representation of the evolution of moving regions. This topic is still an open field for research.

We are particularly interested in the representation of entities and phenomena moving in free space that change their position, shape and extent continuously over time. Therefore, the following subjects are omitted in our discussion, although they might be part of active or intense study and research:

- Network-constrained MOs (Ding, Yang, Güting, & Li, 2015; Hartmut et al., 2006).

- Trajectory databases (TDs) (R. H. Güting, Behr, & Christian, 2012).
- Semantic trajectories (Damiani & Guting, 2014).
- Symbolic trajectories (Damiani, Valdes, & Guting, 2015),
- MOs with different transportation modes, e.g., generic moving objects databases (GMODs) (Xu, 2012).
- MOs with sensors, periodic MOs, the different types of moving objects databases (MODs), e.g., EMOD, NMOD, NMTMOD, GMOD, uncertainty and the management of near future positions, query processing and optimization, storage and indexing structures.

DBMSs are a natural environment to manage data and there is a lot of work on the management of spatiotemporal data in the literature based on DBMSs. In this chapter we presented an overview of the main works that use this approach. We discussed:

- Spatiotemporal databases and its applications, architecture, data models and operations.
- Spatiotemporal query languages.
- Spatiotemporal ORDBMSs.
- The problems of representing the evolution of moving regions.

## A Discrete Data Model for Moving Objects

We start this chapter by giving some initial background that, we think, can help the reader to better understand the context and the objectives of the discrete data model that we present and propose next.

To the best of our knowledge there is not in the literature a data model for moving objects that uses the concept of mesh, obtained by using compatible triangulation methods, to represent moving regions and in the following we give the context necessary to understand the reasons for using this concept to represent them.

Assuming that, we have a set of satellite images representing the evolution of 2 icebergs over time (these images could represent other phenomena as well) and we want to obtain the spatiotemporal data about these 2 moving regions from these images. First we need to get 2D geometric representations of the icebergs' from these images. For this step we use the methods implemented in (Paulo, 2012). We obtain a set of geometries (polygons) that can have a different number of points in their respective boundaries (see Figure 3.1).

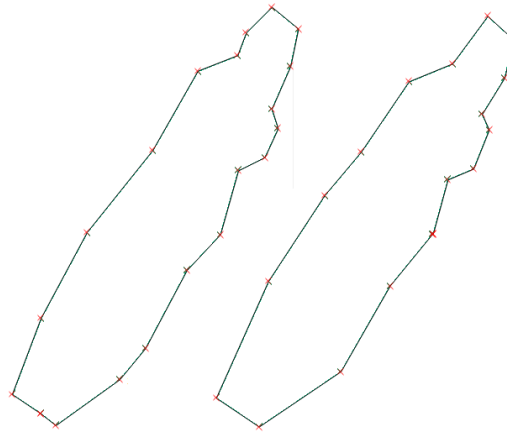


Figure 3.1 - 2 observations of the evolution of an iceberg over time.

We use the *sliced representation* to represent the continuous evolution of these moving regions over time therefore we need to interpolate the evolution of these geometries between its known states. This means that we need a one to one correspondence between the 2 polygons that we want to interpolate. To get this correspondence we use the methods implemented in (Paulo, 2012) and (Mesquita, 2013). These methods take a source polygon,  $P$ , and a target polygon,  $Q$ , and find the correspondence between their boundary points.  $P$  and  $Q$  may have a different number of boundary points so these methods will add cloned or collinear points to  $P$  and  $Q$  boundaries as needed, without changing their shape and extent. Cloned points are points that have the exact position of an existing point and collinear points are points lying on an existing boundary line segment (see Figure 3.2).

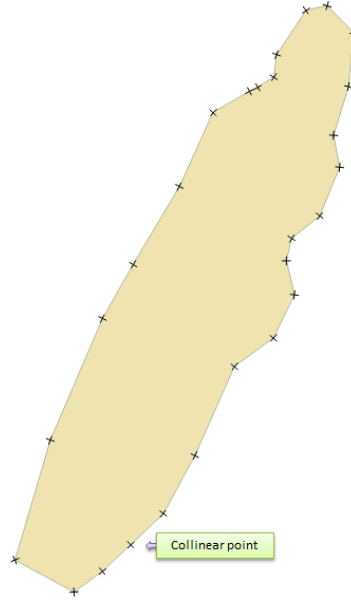


Figure 3.2 - A geometry with collinear points.

Our goal is to obtain a realistic representation of the evolution between these geometries. (Amaral, 2015) shows that the use of compatible triangulation and rigid interpolation methods obtains more realistic representations of the evolution of these geometries when compared to other methods proposed in the literature (Tøssebro & Güting, 2001) and (Mckennney & Frye, 2015).

Therefore we use the methods proposed in (Amaral, 2015) to represent the evolution of these geometries.

The compatible triangulation method used was proposed in (C. Gotsman & Surazhsky, 2004). The method takes 2 polygons, a source and a target polygon,  $P$  and  $Q$ , with a one to one correspondence and generates 2 meshes, see Figure 3.3. The method will also add Steiner points to the interior of the polygon as needed, see Figure 3.4. The authors also propose methods to smooth the meshes, see Figure 3.5.

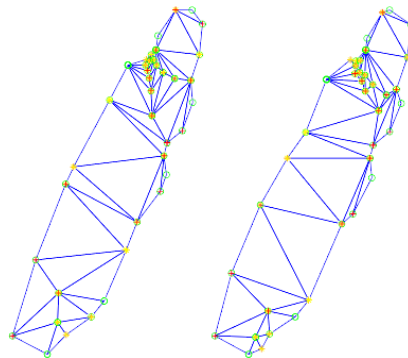


Figure 3.3 - 2 meshes.



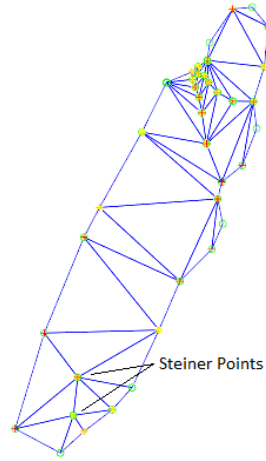


Figure 3.4 - Steiner points.

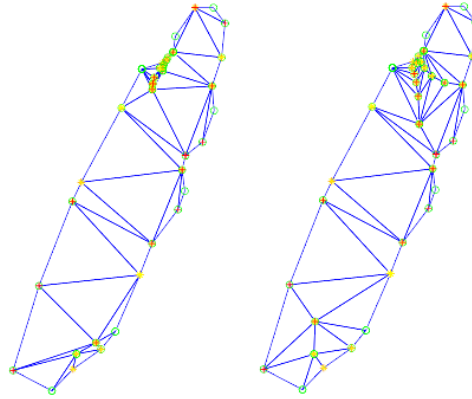


Figure 3.5 - A mesh obtained not using (left) and using (right) the smoothing method.

The rigid interpolation method uses the method proposed in (Baxter et al., 2008). This method uses linear algebra operations, in particular the pseudo inverse of a matrix and the Singular Value Decomposition (SVD) of a matrix, to calculate the interpolation components used to interpolate the meshes. The estimated transformation includes 3 matrices: a transformation matrix, a rotation matrix and a scale matrix.

These methods allow us to obtain spatiotemporal data from real data and represent its evolution over time in a realistic way. Now we want to be able to manipulate, analyse and store this data. This is the main focus of this dissertation. We want to build a framework for moving objects that uses the methods implemented in (Paulo, 2012), (Mesquita, 2013) and (Amaral, 2015) and in this chapter we propose a discrete data model for moving objects, that uses the concept of mesh to represent moving regions, that allows us to achieve that. This concept has its roots in (Amaral, 2015). Moreover, our data model is independent from any specific client using it and that includes DBMSs.

### 3.1 Overview

The main objective of the proposed data model is to define an algebra, i.e., a set of data types and operations, for moving objects using the methods studied in previous works (Amaral, 2015), (Mesquita, 2013) and (Paulo, 2012). Such algebra should have certain properties, in particular, it should be:

- Closed, i.e., it should have a closed system of operations.
- Simple and avoid the proliferation of operations and data types.

We do not define this algebra at an abstract level, i.e., we do not define an abstract data model. Instead we define a discrete data model that has as a reference the abstract model presented in (R. H. Güting et al., 2000) and the discrete model presented in (Forlizzi et al., 2000). We focus on moving regions and on a subset of the operations presented in (R. H. Güting et al., 2000) that we find representative and interesting.

Our model defines moving regions using compatible triangulation methods and thus requires a new spatial type called *mesh*. This is not the approach of the models that it has as a reference.

The data types in our discrete model are divided into BASE types, TIME types, SPATIAL types, i.e., OGC-compliant spatial types, and the new spatial type *mesh*, INTERVAL, PERIOD, FUNCTION, UNIT and MOVING types. Table 3.1 shows the signature of these data types. TIME types represent time, INTERVAL types represent intervals of values, e.g., an interval of time in the form  $[\text{instant}_i, \text{instant}_j]$ , PERIOD types represent sets of intervals, SPATIAL types represent geometries in the 2D Cartesian plane, FUNCTION types represent functions in the mathematical sense, e.g.,  $ax + b$ , UNIT types represent the evolution of moving objects and the changes of bool and real values during an interval of time and MOVING types represent moving objects and values that change continuously (*mreal*) and in discrete steps (*mbool*) over time.

The MOVING types are represented using the *sliced representation* presented in (Forlizzi et al., 2000) and we do not consider moving lines, geometries with holes and collections.

|  |            |  |
|--|------------|--|
|  | → BASE     | <i>int, real, bool</i>   |
|  | → SPATIAL  | <i>point, linestring, polygon, multipoint, multilinestring, multipolygon, geometrycollection, mesh</i> |
|  | → TIME     | <i>instant</i>   |
| $\text{BASE} \setminus \{\text{bool}\} \cup \text{TIME}$ | → INTERVAL | <i>interval</i>  |
| INTERVAL   | → PERIOD   | <i>period</i>  |
| $\text{BASE} \cup \text{SPATIAL}$                        | → UNIT     | <i>ubool, ureal, upoint, umesh</i>   |
|  | → FUNCTION | <i>function</i>  |
| UNIT   | → MOVING   | <i>mbool, mreal, mpoint, mmesh</i>   |

Table 3.1 - The signature of our discrete data model for moving objects.

In Table 3.1 the words in capital letters can be thought of as sets, e.g., MOVING is the set  $\{mbool, mreal, mpoint, mmesh\}$ , therefore when we say the MOVING types we mean the *mbool*, *mreal*, *mpoint* and *mmesh* types. The semantics of  $BASE \setminus \{bool\} \cup TIME \rightarrow INTERVAL$  is that we can have intervals of *int*, *real*, and *instant* but not an interval of *bool* (an interval of *bool* does not make sense).

In the next sections of this chapter we define, present and discuss our proposed discrete data model for moving objects, in particular, for moving regions. We start by defining its data types. Then we define the set of operations over those data types.

### 3.2 Notation and Assumptions

The notation used in this chapter follows the convention shown in Table 3.2.

| Symbol             | Description   |
|--------------------|---|
| $t$                | An <i>instant</i> .   |
| $I$                | An interval of time, i.e., <i>interval(instant)</i> .   |
| $P$                | A <i>period</i> .   |
| $s$                | A spatial type, i.e., $s \in SPATIAL$ .   |
| $\mu, \varphi$     | 2 moving types, i.e., $\mu, \varphi \in MOVING$ .   |
| $ a $              | Gives the number of elements of $a$ .   |
| <u><i>bool</i></u> | Represents the underlying programming language type used to represent the values $\{true, false\}$ or equivalently $\{1, 0\}$ . |

Table 3.2 - Notation.

Our discrete data model is built on top of the following assumptions:

- Collections and lines are not considered.
- The model only allows the use of polygons without holes.
- Time advances from past to future in a totally ordered form. Granularity, non-linearity of time and considerations about time being isomorphic to the integer, the rational or the real numbers are left for future analysis. Hermes (Nikos Pelekis et al., 2010) is a good starting point for such discussion.
- SPATIAL types are represented in the two-dimensional (2D) Cartesian plane and their representation is based on linear approximations.

### 3.3 Type System Data Types

#### 3.3.1 Base Types and Time Types

The BASE types: *int*, *real*, *bool*, and the TIME types: *instant*, are defined in the same way as in (Forlizzi et al., 2000). These types map to the underlying programming language types that represent the integer numbers (*int*), the real numbers (*real* and *instant*) and the values in the set {true, false} (*bool*).

#### 3.3.2 Interval and Period Types

Type *interval* follows the definition for Interval and type *period* follows the definition for IntervalSet given in (Forlizzi et al., 2000).

#### 3.3.3 Spatial types

The spatial types are: *point*, *linestring*, *polygon*, *multipoint*, *multilinestring*, *multipolygon*, *geometrycollection* and *mesh*. With the exception of *mesh* they all conform to the Open Geospatial Consortium's (OGC) OpenGIS Specifications (OGC Simple Features, 2016) and will not be defined here.

A *mesh* type represents a region with the characteristics supported by the model. It is essentially a triangulated polygon. It has a boundary, Steiner points, and a set of non-overlapping triangles, see Figure 3.6.

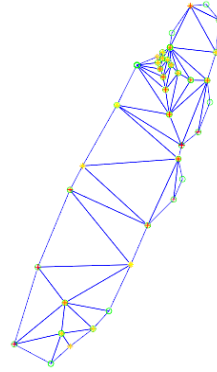


Figure 3.6 - A mesh of an iceberg, obtained using our framework for moving objects.

To define the *mesh* type we start by redefining the set of all line segments, *Seg*, defined in (Forlizzi et al., 2000) to allow a line segment to degenerate to a point. We need to do this because the boundary of a *mesh* can have duplicate points, i.e., points with exactly the same x and y coordinates.

Let  $\text{Seg}^+$  be the set of all line segments. Then:

$$\text{Seg}^+ = \{(u, v) \mid u, v \in \text{point}, u \leq v\}$$

As a consequence, we need to redefine a cycle, also defined in (Forlizzi et al., 2000). A cycle is a simple polygon that is defined using *Seg*. Our definition of a cycle uses  $\text{Seg}^+$  as follows:

$$\text{Cycle}^+ = \{S \subset \text{Seg}^+ \mid |S| \geq 3; \text{ such that}$$

$$(i) \quad \forall s_i, s_j \in S : (s_i \neq s_j \vee (s_i = s_j \wedge \text{p-degenerated}(s_i))) \Rightarrow (\neg \text{p-intersect}(s_i, s_j) \wedge \neg \text{touch}(s_i, s_j))$$

$$(ii) \quad \forall p \in \text{points}(S) : \text{card}(p, S) = \varphi$$

Where:  $p\text{-intersect}(s_i, s_j)$ ,  $\text{touch}(s_i, s_j)$ ,  $\text{points}(S)$  and  $\text{card}(p, S)$  have the meaning given in (Forlizzi et al., 2000), i.e., 2 segments  $p\text{-intersect}$  if they intersect in their interior (in a point other than an end point), 2 segments  $\text{touch}$  if one end point of one of the segments is in the interior of the other one,  $\text{points}(S)$  is the set of all the end points of the segments in  $S$ ,  $\text{card}(p, S)$  is a function that computes how often point  $p$  occurs in  $S$ ,  $p\text{-degenerated}(s_i)$  means that  $s_i$  is a line segment degenerated to a point and  $\varphi$  is defined as  $\varphi = m + 2$ ,  $m \geq 0$ ,  $m$  is the number of duplicates of  $p$ .

(i) and (ii) have the same meaning as in (Forlizzi et al., 2000). That is, (i) means that no segments intersect properly and (ii) means that each end point occurs in exactly  $\varphi$  segments.

#### Notes:

- In (i) if  $s_i = s_j$  then  $p\text{-degenerated}(s_i)$  and  $p\text{-degenerated}(s_j)$ , so we only show one of them. That is, if  $s_i = s_j$  they are both  $p\text{-degenerated}$ .
- 2 line segments can be collinear.
- Because we allow degenerated line segments, a line segment can meet more than 2 other line segments.
- Collinear and point-degenerated line segments are *mesh* properties that deserve a more careful analysis in the future. Intuitively, they may give rise to geometries with bad or undesirable properties.

We need to define  $\Delta$  before defining *mesh*.

Let *Triangle* be the set of all triangles.

$$\text{Triangle} = \{(p_1, p_2, p_3) \mid p_1, p_2, p_3 \in \text{point}\}$$

We put no restrictions on the elements of a triangle. As a consequence, a triangle is allowed to degenerate to a line and to a point.

Then  $\Delta$  is defined as follows:

$$\Delta = \{T \subset \text{Triangle} \mid |T| \geq 1, \text{ such that}$$

$$(i) \quad \forall \text{tri}_i, \text{tri}_j \in T : \text{tri}_i \neq \text{tri}_j \Rightarrow \neg \text{overlap}(\text{tri}_i, \text{tri}_j)\}$$

Where  $\neg \text{overlap}(\text{tri}_i, \text{tri}_j)$  has the semantics that 2 triangles can share boundary points but their interiors cannot intersect.

Finally, we can define a *mesh*. Formally, *Mesh*, the set of all meshes, is defined as follows:

$$\text{Mesh} = \{(b, o, \Delta) \mid b \in \text{Cycle}^+, o = \text{ul}(\text{point}) \wedge |o| \geq 0, |\Delta| > 0, \text{ such that}$$

$$(i) \quad \text{covers}(b, o)\}$$

Where  $\text{covers}(b, o)$  has the semantics that every point in  $o$  is a point of the interior or boundary of  $b$ ,  $|o|$  is the number of Steiner points of the *mesh* and  $\text{ul}$  represents an unordered list. We allow a Steiner point to be exactly on the mesh's boundary and this is a discussion for future work.

Table 3.3 gives a description of the components of a *mesh*.

| Component | Description  |
|-----------|--|
| b         | The mesh's boundary is a cycle.  |
| o         | The mesh's Steiner points are represented using an unordered list of n points, $ul(point)$ . In the current definition there is no restriction on the uniqueness of a point. |
| $\Delta$  | The <i>mesh</i> triangles. This is a finite set of non-overlapping triangles. A <i>mesh</i> has at least one triangle.   |

Table 3.3 - Type *mesh* components.

Type *mesh* is used to represent mesh objects (see Figure 3.6).

Finally, we need to define the concept of mesh continuity, i.e., the conditions in which 2 meshes are considered to be continuous. This concept is needed to establish continuity for the *umesh* type presented in section 3.3.6. Given  $m_i, m_j \in Mesh$ , then we define mesh continuity as:

$$distance(m_i, m_j) \leq \xi_p \quad (1) \quad \wedge \quad \frac{m_i \cap m_j}{m_i \cup m_j} \leq \delta_s \quad (2)$$

Where  $distance(m_i, m_j)$  is the distance between the centroids of the 2 meshes  $m_i$  and  $m_j$  and  $\xi_p$  and  $\delta_s$  are 2 constant values. 2 meshes are continuous if they are very close to each other or as close as we want (1) and if their shapes and extents are similar (2). This concept was not strictly established.

### 3.3.4 Sliced Representation for Moving Objects

We use the *sliced representation* and the temporal unit concepts, proposed and defined in (Forlizzi et al., 2000), to represent the MOVING types: *mbool*, *mreal*, *mpoint* and *mmesh*. That is, a MOVING type is represented as a set of temporal units or slices. Notice that, *mbool* represents a value that changes in discrete steps over time.

Briefly, a temporal unit, or a unit, seen as a generic concept, represents the evolution of a value of a given type  $\alpha$  over an interval of time, i.e.:

$$Unit(\alpha) = I \times \alpha, \forall \alpha \in \text{BASE} \cup \text{SPATIAL}$$

Then, a unit  $u_\alpha \in Unit(\alpha)$  is defined as a pair of a unit interval and a unit function:

$$u_\alpha = (I, v(t)), v(t) \in UnitFunction$$

Where  $I$  represents the unit interval,  $v(t)$  represents the evolution of a value of a given type  $\alpha$  over  $I$  and  $(Unit(\alpha), <)$  is a set with a total order, i.e., we can establish a notion of order between units.

*UnitFunction* represents a function as in the mathematical sense and it is defined as follows:

$$UnitFunction = \{(a, b, c, type) \mid a, b, c \in \text{real}, type \in \{\text{CONST}, \text{LINEAR}, \text{POLN\_2}\}\}$$

Where:

- $a, b, c$  are the coefficients of a function:  $f(t) = a^2t + bt + c$ .
- *type* is the type of function: CONST:  $f(t) = c$ , LINEAR:  $f(t) = bt + c$ , POLN\_2:  $f(t) = a^2t + bt + c$ .

These concepts are generic enough to allow the type system to be extended with new UNIT types and types of functions.

In the next subsections we define the UNIT types of our discrete model: *ubool*, *ureal*, *upoint* and *umesh*. They are specializations of this generic concept.

### 3.3.5 Temporal Units for Base Data Types

Type *ubool* represents the evolution of a *bool* type during a given interval of time.

The set for *ubool* is defined as:

$$UBool = \{(I, v) \mid v \in \underline{bool}\}$$

Where:

- $I$  is the interval of time in which the unit is defined.
- $v$  is a constant value assumed during  $I$ .

Type *ureal* represents the evolution of a *real* type during a given interval of time.

The set for *ureal* is defined as:

$$UReal = \{(I, v(t)) \mid v(t) \in UnitFunction\}$$

Where:

- $I$  is the interval of time in which the unit is defined.
- $v(t)$  is a function that represents the evolution of the *real* type during  $I$ .

### 3.3.6 Temporal Units for Spatial Data Types

Type *upoint* represents the evolution of a *point* type, i.e., the evolution of its x and y coordinates, in the 2D Cartesian plane during a given interval of time.

The set for *upoint* is defined as:

$$UPoint = \{(I, x(t), y(t)) \mid x(t), y(t) \in UnitFunction\}$$

Where:

- $I$  is the interval of time in which the unit is defined.
- $x(t)$  is a function that represents the evolution of the point's x coordinate during  $I$ .
- $y(t)$  is a function that represents the evolution of the point's y coordinate during  $I$ .

Type *umesh* represents the evolution of a *mesh* type in the 2D Cartesian plane during a given interval of time.

The set for *umesh* is defined as:

$$UMesh = \{(I, P, Q, IC) \mid P, Q \in mesh, \text{ such that}$$

- (i)  $\text{correspondence}(P, Q)\}$

Where:

- $I$  is the interval of time in which the unit is defined.
- $P$  and  $Q$  represent the original source and target meshes at the begin and end instants of  $I$ .
- $IC$  represents the interpolation components, computed by the interpolation method, used to interpolate the *mesh* during  $I$ . They will be presented and discussed in chapter 4 that deals with implementation details. *The umesh* type is independent of the interpolation method details.
- $\text{correspondence}(P, Q)$  means that  $P$  and  $Q$  must have an implicit one to one correspondence. This is a restriction imposed by the triangulation method.

### 3.3.7 Moving Types

The definition and constraints for the MOVING types follow the ones given for the mapping type constructor in (Forlizzi et al., 2000) and we add the following:  $|\varphi| \geq 0$ ,  $|\varphi|$  is the number of units of the moving type  $\varphi$ . If  $|\varphi| = 0$  we say that  $\varphi$  is empty. That is, we allow a moving type to be empty.

With the exception of *mbool*<sup>1</sup>, when 2 units  $u_\alpha$  and  $v_\alpha$ ,  $u_\alpha \neq v_\alpha$ , of type  $\alpha$  meet at one of the end points of their respective unit intervals, i.e.,  $u_\alpha.I.b = v_\alpha.I.e \vee u_\alpha.I.e = v_\alpha.I.b$ , we need to enforce unit continuity.  $I.b$  and  $I.e$  are the begin and end instants of interval  $I$ , respectively.  $u_\alpha.I$  represents the unit interval of unit  $u_\alpha$ , i.e., the time interval in which unit  $u_\alpha$  is defined.

As a generic concept, we consider that 2 units of type  $\alpha$  are continuous if the distance between their values at *instant*  $t$  where they meet, is bellow some  $\xi$ . That is, given  $u_\alpha, v_\alpha \in \text{Unit}(\alpha)$  and  $u(t)$ , a function that gives the discrete value of some unit of type  $\alpha$  at *instant*  $t$ :

$$\text{distance}(u_\alpha.u(t), v_\alpha.u(t)) \leq \xi \Leftrightarrow \text{continuous}(u_\alpha, v_\alpha)$$

In the present definition we do not establish a notion of continuity for *ureal*, see section 6.4.2 for more details on this. Type *upoint* continuity is checked by computing the distance between the points' positions. For type *umesh* we use the mesh continuity concept defined in section 3.3.3.

## 3.4 Operations on Moving Types

This section presents a subset of the operations proposed in (R. H. Güting et al., 2000). We chose the ones that we found more representative and interesting to include in our model, to prove that our framework for moving objects can provide operations from all the classes of operations proposed in the literature. We chose some operations because they can be used as a basis for implementing other operations and to be able to use and test all the MOVING types that our data model defines, see Table 3.4.

| Class of Operation | Operation |
|--------------------|-----------|
|--------------------|-----------|

<sup>1</sup> *mbool* represents a data type that changes in discrete steps over time.



|                                   |                              |
|-----------------------------------|------------------------------|
| Predicates                        | equals, intersects           |
| Set Operations                    | intersection, union          |
| Numeric                           | area                         |
| Projection to Domain and Range    | deftime, traversed           |
| Interaction with Domain and Range | atinstant, atperiod, present |
| Constructors                      | unit, moving                 |

Table 3.4 - The set of operations on MOVING types in our discrete model.

Note:

- The use of  $\varphi$  in operation signatures as in:  $\varphi \times \varphi \rightarrow Type$  and  $\varphi \times Type \rightarrow \varphi$ , means that the operations are applicable to MOVING types of the same type.
- The use of  $\varphi$  and  $\mu$  in operation signatures as in:  $\varphi \times \mu \rightarrow Type$  and  $\varphi \times Type \rightarrow \mu$ , means that the operations are applicable to MOVING types of different types.

### 3.4.1 Predicates

Predicates are operations that return {true, false} values related to topological and other relationships between MOVING types. See Table 3.5.

| Operation  | Signature                                      | Semantics   |
|------------|--|---|
| equals     | $\varphi \times \varphi \rightarrow bool$      | Checks if 2 MOVING types are equal.                             |
| intersects | $mmesh \times mmesh \rightarrow mbool$         | Checks if 2 <i>mmesh</i> types intersect.                       |
|            | $mmesh \times mpoint \rightarrow mbool$        | Checks if a <i>mmesh</i> and a <i>mpoint</i> type intersect.    |
|            | $mmesh \times mmesh \times t \rightarrow bool$ | Checks if 2 <i>mmesh</i> types intersect at an <i>instant</i> . |
|            | $mpoint \times mpoint \rightarrow mbool$       | Checks if 2 <i>mpoint</i> types intersect.                      |
|            | $mmesh \times s \rightarrow mbool$             | Checks if a <i>mmesh</i> and a SPATIAL type intersect.          |
|            | $mpoint \times s \rightarrow mbool$            | Checks if a <i>mpoint</i> and a SPATIAL type intersect.         |

Table 3.5 - Predicate operations.

### 3.4.2 Set Operations

The set operations that we considered are shown in Table 3.6.

| Operation    | Signature                                 | Semantics  |
|--------------|---|--|
| intersection | $mmesh \times point \rightarrow mpoint$   | Computes the intersection of the arguments.      |
|              | $mmesh \times polygon \rightarrow mmesh$  |  |
|              | $mmesh \times mmesh \rightarrow mmesh$    |  |
|              | $mmesh \times mmesh \times t \rightarrow$ | Computes the intersection at an <i>instant</i> . |

|       |  |                                      |
|-------|--|--------------------------------------|
|       | $geometrycollection \cup polygon$      |                                      |
| union | $mmesh \times mmesh \rightarrow mmesh$ | Computes the union of the arguments. |

Table 3.6 - Set operations.

### 3.4.3 Numeric

Operations that compute some numeric value, e.g., the area or the perimeter of a *mmesh*. See Table 3.7.

| Operation | Signature                          | Semantics  |
|-----------|------------------------------------|--|
| area      | $mmesh \times t \rightarrow real$  | Returns the area of a <i>mmesh</i> type at an <i>instant</i> . |
|           | $mmesh \times P \rightarrow mreal$ | Returns the area of a <i>mmesh</i> type at a <i>period</i> .   |

Table 3.7 - Numeric operations.

### 3.4.4 Projection to Domain and Range

Projection of MOVING types to the time (range) and spatial domains (domain). See Table 3.8.

| Operation | Signature                              | Semantics   |
|-----------|--|---|
| deftime   | $\varphi \rightarrow P$                | Returns the <i>period</i> in which a MOVING type is defined.              |
| traversed | $mmesh \rightarrow geometrycollection$ | Computes the projection of a <i>mmesh</i> type in the 2D Cartesian plane. |

Table 3.8 - Projection of MOVING types to Domain and Range.

### 3.4.5 Interaction with Domain and Range

Restriction of MOVING types to the time (range) and spatial domains (domain). See Table 3.9.

| Operation | Signature                              | Semantics  |
|-----------|--|--|
| atinstant | $mmesh \times t \rightarrow polygon$   | Restricts the MOVING type given as an argument to a specified <i>instant</i> . |
|           | $mpoint \times t \rightarrow point$    |  |
|           | $mreal \times t \rightarrow real$      |  |
|           | $mbool \times t \rightarrow bool$      |  |
| atperiod  | $\varphi \times P \rightarrow \varphi$ | Restricts the MOVING type given as an argument to a specified <i>period</i> .  |
| present   | $\varphi \times t \rightarrow bool$    | Checks whether the MOVING type exists at a specified <i>instant</i> .          |
|           | $\varphi \times P \rightarrow mbool$   | Checks whether the MOVING type exists at a specified <i>period</i> .           |

Table 3.9 - Restriction of MOVING types to the time and spatial domains.

### 3.4.6 Constructors

Operations to construct UNIT and MOVING types. See Table 3.10.

| Constructor | Signature  | Semantics  |
|-------------|--|--|
| UnitBool    | $I \times bool \rightarrow ubool$  | Constructs an <i>ubool</i> type.   |
| UnitReal    | $I \times v_b \times v_e \times type \rightarrow ureal$                                      | Constructs a <i>ureal</i> type. <ul style="list-style-type: none"> <li><math>v_b, v_e \in real</math>, begin and end values of the unit's value.</li> <li><math>type \in int</math>, type of function to interpolate the unit's value during <math>I</math>.</li> </ul>  |
| UnitPoint   | $I \times x_b \times y_b \times x_e \times y_e \times typeX \times typeY \rightarrow upoint$ | Constructs an <i>upoint</i> type. <ul style="list-style-type: none"> <li><math>x_b, y_b \in real</math>, initial position of the point.</li> <li><math>x_e, y_e \in real</math>, end position of the point.</li> <li><math>typeX \in int</math>, type of function to interpolate the x coordinate.</li> <li><math>typeY \in int</math>, type of function to interpolate the y coordinate.</li> </ul> |
| UnitMesh    | $I \times P \times Q \times IC \rightarrow umesh$  | Constructs an <i>umesh</i> type. <ul style="list-style-type: none"> <li>See section 3.3.6.</li> </ul>  |
| MovingBool  | $\rightarrow mbool$  | Constructs an empty MOVING type.   |
| MovingReal  | $\rightarrow mreal$  |  |
| MovingPoint | $\rightarrow mpoint$   |  |
| MovingMesh  | $\rightarrow mmesh$  |  |

Table 3.10 - Constructors.

## 3.5 Summary

In this chapter we proposed and presented a discrete data model for moving objects that uses the concept of mesh to represent moving regions. This model:

- Is independent from any specific client or application using it, and that includes DBMSs.
- Defines a new spatial type called *mesh* used to implement the concept of mesh that the model uses.
- Is independent from the compatible triangulation and interpolation methods used as long as they can work together.
- Has limitations and defines only a small set of the spatiotemporal operations proposed in the literature, e.g., we do not consider lines, geometries with holes and collections.

The MOVING types are represented using the *sliced representation* proposed in (Forlizzi et al., 2000) and the data types are divided into: BASE types, TIME types, SPATIAL types, i.e., OGC-compliant spatial types and the new spatial type *mesh*, INTERVAL, PERIOD, FUNCTION, UNIT and MOVING types.

## Implementation

In the next chapter we discuss the implementation details of our framework for moving objects and the implementation of a spatiotemporal extension for PostgreSQL that uses it. We start with an overview of: our main goals, what was implemented, what technologies and architecture were used. Then we discuss implementation specific needs and the technologies and architectures that we considered using, their advantages and disadvantages. We discuss the implementation details of SPTMesh and MeshGIS (see Table 4.1 for a description of these components): its dependencies, data structures, operations, architecture, usage and installation and implementation issues.

We present a ‘standard’ way of expressing spatiotemporal objects called Spatiotemporal Well-Known Text (STWKT).

Finally, we discuss the structure of the source code of SPTMesh and MeshGIS and the extension of SPTMesh with new triangulation and smoothing methods.

### 4.1 Overview

In this chapter, when we say evolution we mean: the continuous changes in position or shape and extent over time.

After establishing our discrete data model for moving objects, presented in chapter 3, we started the implementation phase. Our main goals were:

- Implement the algebra proposed in chapter 3, i.e., the data types and operations defined by our discrete data model, in a framework that is not dependent on a specific client or architecture using it.
- Manipulate and analyse 2D geometries in the 2D Cartesian plane<sup>2</sup>.
- Integrate the triangulation (C. Gotsman & Surazhsky, 2004), smoothing (C. Gotsman & Surazhsky, 2004) and interpolation (Alexa et al., 2000; Baxter et al., 2008) methods implemented in (Amaral, 2015) as part of our framework.
- Build a spatiotemporal database extension as a proof of concept that our framework works with real applications using real data.
- Build a framework that can be used as a reference for future work and investigation in this area.

---

<sup>2</sup> We want to use the Cartesian coordinate system.

We analysed technologies and architectures to support our implementation, see section 4.2 and implemented 2 components, called: SPTMesh and MeshGIS. See Table 4.1 for details.

| Component | Description   |
|-----------|---|
| SPTMesh   | <p>A framework for moving objects, implemented as a C++ library with a C API. It implements:</p> <ul style="list-style-type: none"> <li>• The data model proposed in chapter 3.</li> <li>• The triangulation, smoothing and interpolation methods implemented in (Amaral, 2015).</li> </ul> |
| MeshGIS   | <p>A spatiotemporal database extension built on top of PostgreSQL, implemented as a C library.</p> <p>We also provide a binding of the MeshGIS functions and data structures to Structured Query Language (SQL) functions and types.</p>  |

Table 4.1 - Implemented components.

The implementation details of these components are discussed in sections 4.3 and 4.4.

Table 4.2 and Table 4.3, below, present the technologies and the development environment setup that we used.

| Component | Technologies Used     |  |
|-----------|-----------------------|--|
| SPTMesh   | Programming Languages | C++11 and C.   |
|           | External Dependencies | Armadillo, BLAS, LAPACK, GEOS.   |
|           |                       |  |
|           | <b>Dependency</b>     | <b>Usage</b>   |
|           | Armadillo             | Linear algebra operations.   |
|           | BLAS, LAPACK          | Armadillo dependencies to provide various matrix decompositions, e.g., SVD.        |
| MeshGIS   | GEOS                  | C++ library for manipulation and analysis of 2D geometries and spatial operations. |
|           | Programming Languages | C.   |
|           | External Dependencies | PostgreSQL, SPTMesh.   |
|           | <b>Dependency</b>     | <b>Usage</b>   |

|  |            |   |
|--|------------|---|
|  | PostgreSQL | MeshGIS is built on top of PostgreSQL. PostgreSQL makes it possible to, among other things: <ul style="list-style-type: none"> <li>• Store moving objects in a database.</li> <li>• Analyse and manipulate moving objects using SQL.</li> </ul> |
|  | SPTMesh    | C++ library for manipulation and analysis of moving objects.  |

Table 4.2 - Technologies used to implement SPTMesh and MeshGIS.

| Application   | Usage  |
|---|--|
| Windows 8.1 Pro 32-Bit on a VMware virtual machine. | Operating System.  |
| PostgreSQL 9.4                                      | Database Management System that supports MeshGIS.  |
| PostGIS 2.1.8                                       | Store and perform spatial operations on MeshGIS spatial results.                                     |
| Microsoft Visual Studio Community 2015              | Integrated Development Environment (IDE) used to implement SPTMesh and MeshGIS.                      |
| QGIS 2.16.1   | Visualization of geometries and operation results stored in PostgreSQL.                              |
| Octave 4.0.1  | Visualization of geometries and operation results, e.g., triangulation, smoothing and interpolation. |

Table 4.3 - Development environment setup.

Figure 4.1 shows an overall overview of the architecture of a system using SPTMesh, MeshGIS and possibly other external systems, e.g., PostGIS. PostGIS is used to emphasize the fact that we can do composition of functions to provide further value, using functions provided by MeshGIS and PostGIS. The 2 systems can exchange spatial data using the Well-Known Text (WKT) standard form for expressing spatial objects given by the OpenGIS Consortium (OGC) standard<sup>3</sup>.

<sup>3</sup> <http://www.opengeospatial.org/standards/wkt-crs>

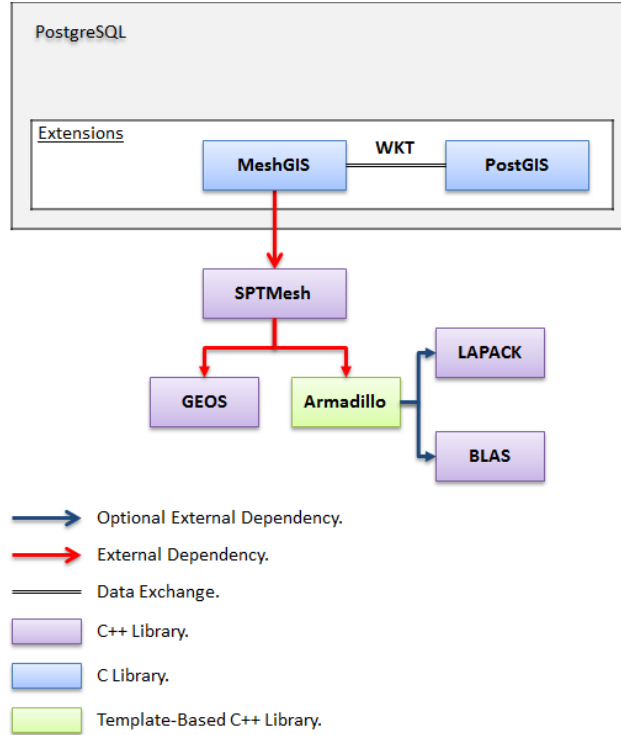


Figure 4.1 - Overall overview of the architecture of a system that uses MeshGIS and SPTMesh.

It is important to mention that MeshGIS and PostGIS do not communicate directly with each other. They communicate using the SQL functions that bind to the functions that they provide.

In the next sections we discuss implementation in more detail.

## 4.2 Technologies and Architectures

In this section we discuss the technologies and architectures that we considered using: their advantages and disadvantages, and the reasons behind some of our choices.

Our implementation has specific needs that we have to consider:

- The triangulation, smoothing and interpolation methods that we use were initially implemented in Matlab and use linear algebra operations, in particular to compute the Moore-Penrose pseudo-inverse of a matrix and to perform the Singular Value Decomposition (SVD) of a matrix.
- Our discrete data model uses the SPATIAL types: point, linestring, polygon, multipoint, multilinestring, multipolygon and geometrycollection. They are already well-established and various implementations are available. This suggests that they can be provided by an external dependency.
- The architecture of our framework for moving objects, i.e., SPTMesh, should be independent from any specific client using it, this includes database systems, and it should be portable.

This project started from scratch so there were no more initial restrictions other than the specific needs that were previously identified.



### 4.2.1 To Implement a Framework for Moving Objects

We considered several programming languages to implement SPTMesh: java, python and C++. They all provide tools to manipulate and analyse 2D geometries in the Cartesian plane and to perform linear algebra operations. Python, in particular, provides some of these tools as wrappers for C++ libraries, e.g., Shapely<sup>4</sup>.

We looked for and analysed well-known and well-established frameworks for spatial and spatiotemporal objects, such as: Hermes<sup>5</sup> and PostGIS<sup>6</sup>. PostGIS, in particular, uses GEOS<sup>7</sup>, a C++ library, for spatial operations on geometries on the Cartesian plane. And this is particularly relevant for us.

Then, we compared implementation alternatives for SPTMesh. Given the sets: {C++, Python}, that represents the set of programming languages to implement SPTMesh, and {GEOS, Shapely, PostGIS}, that represents the set of tools to perform spatial operations on geometries on the Cartesian plane, we have at least 3 trivial alternatives:

- Scenario 1. Using C++ and GEOS:

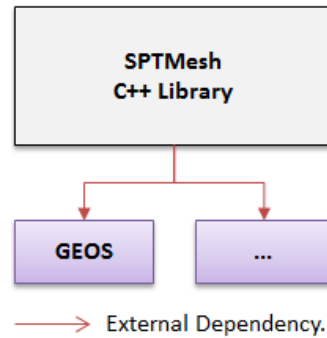


Figure 4.2 - SPTMesh architecture using C++ and GEOS.

This is a simple and elegant solution.

- Scenario 2. Using Python and Shapely:

---

<sup>4</sup> <http://toblerity.org/shapely/project.html> (Shapely is a wrapper for the GEOS library.)

<sup>5</sup> [https://hermes-mod.java.net/manual.html#Architecture\\_Map](https://hermes-mod.java.net/manual.html#Architecture_Map)

<sup>6</sup> <http://postgis.net/>

<sup>7</sup> <http://geos.osgeo.org/>

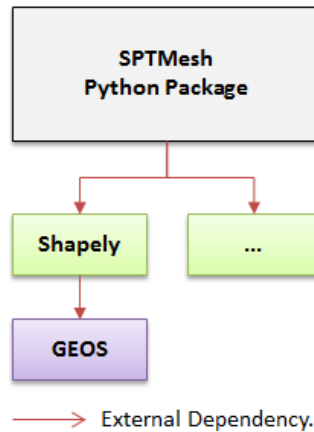


Figure 4.3 - SPTMesh architecture using Python and Shapely.

This solution is similar to the previous one but, we can have additional levels of indirection because some Python tools are actually wrappers for other libraries, e.g., Shapely is a wrapper for the GEOS library.

- Scenario 3. Using C++ or Python and PostGIS:

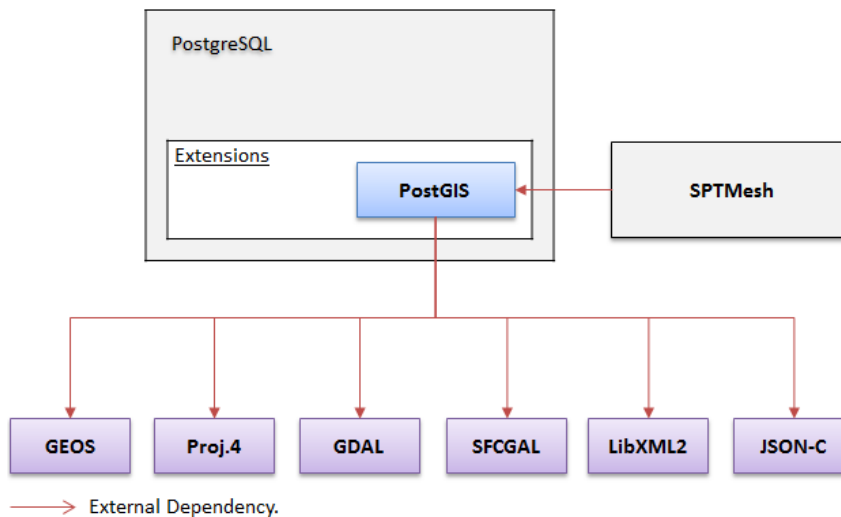


Figure 4.4 - SPTMesh architecture using PostGIS.

This scenario, if feasible at all, is not an interesting solution. The strong coupling between PostGIS and PostgreSQL is automatically inherited by SPTMesh compromising the independence and portability properties of any architecture.

We also analysed the GEOS C++ library architecture, shown in Figure 4.5.

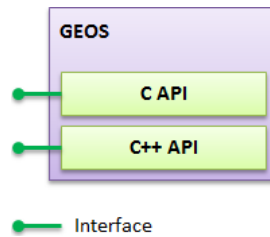


Figure 4.5 - GEOS Architecture.

Briefly, GEOS is a project of the Open Source Geospatial Foundation (OSGeo<sup>8</sup>). It is a C++ library with a C API used by projects such as PostGIS and Autodesk MapGuide Enterprise, among others. GEOS also provides all the SPATIAL data types, with the exception of the *mesh* type, that our discrete data model uses. That is, GEOS is a well-known, well-established project used to analyse and manipulate spatial objects.

There are several libraries for linear algebra available, such as: Armadillo, Eigen<sup>9</sup>, Eigen do Better<sup>10</sup>, LAPACK<sup>11</sup> and OpenBLAS<sup>12</sup>. In particular, Eigen, Eigen do Better and Armadillo are template-based C++ libraries and this is a very interesting property.

Eigen is a well-known, well-established library. It has no dependencies other than the C++ standard library. However, Eigen does not provide a method to compute the pseudo-inverse of a matrix and the provided SVD of a matrix does not meet our needs.

Eigen do Better is an improved version of the Eigen library. It does provide a method to compute the pseudo-inverse of a matrix but the ‘SVD problem’ remains unsolved.

Armadillo does provide a method to compute the pseudo-inverse of a matrix and the provided SVD of a matrix does meet our needs. Unlike Eigen, Armadillo provides several matrix decompositions using external dependencies. One of these decompositions is the SVD of a matrix. Therefore, we use LAPACK and OpenBLAS as Armadillo external dependencies.

After this analysis, we established that:

- The SPTMesh architecture follows the architecture presented in Scenario 1 and the GEOS architecture. Therefore, SPTMesh provides a C API on top of a C++ implementation and that should be the interface used by applications linking to it so that these applications still work without relinking when the framework is upgraded.
- SPTMesh uses GEOS for manipulation and analysis of 2D geometries in the Cartesian plane, i.e., for spatial operations. GEOS does not provide any support for the *mesh* type.
- SPTMesh uses Armadillo<sup>13</sup> (Sanderson & Curtin, 2016) for linear algebra operations.

<sup>8</sup> <http://www.osgeo.org/>

<sup>9</sup> <http://eigen.tuxfamily.org/>

<sup>10</sup> <http://eigendobetter.com/>

<sup>11</sup> <http://www.netlib.org/lapack/>

<sup>12</sup> <http://www.openblas.net/>

<sup>13</sup> <http://arma.sourceforge.net>

SPTMesh implementation details are discussed in section 4.3.

## 4.2.2 To implement a Spatiotemporal Database Extension

To implement MeshGIS we considered 2 DBMSs: Oracle and PostgreSQL. There is a lot of work done on spatiotemporal database extensions using Oracle in the literature. Not so many using PostgreSQL. Noticeably, Hermes (Nikos Pelekis et al., 2010) is a spatiotemporal database extension built on top of Oracle and PostgreSQL.

We decided to use PostgreSQL because it is a well-established project, allows for the creation of user extensions with user data types and operations, it is a powerful modern database engine, it is free, easy to use and install, runs on several operating systems, it has a powerful spatial extension, i.e., PostGIS, and it was used successfully to implement Hermes.

The PostgreSQL backend is implemented in C. It might, however, be possible to load functions written in other languages, namely: C++, FORTRAN or Pascal, into PostgreSQL and we can write PostgreSQL extensions using C++, if certain guidelines<sup>14</sup> are followed. PostgreSQL also supports procedural languages<sup>15</sup>, e.g., PL/Python that allows PostgreSQL to use functions written using Python, but they have several significant limitations<sup>16</sup>. Following the examples of PostGIS and Hermes and in order to avoid unnecessary complexity we decided to implement MeshGIS as a C library. Furthermore, MeshGIS follows the architecture of PostGIS and its implementation details are discussed in section 4.4.

## 4.3 SPTMesh - A Framework for Moving Objects

SPTMesh is a framework for moving objects implemented as a C++ library with a C API that follows the architecture of GEOS. It implements:

- The data types and part of the operations proposed in the data model for moving objects presented in chapter 3.
- The triangulation, smoothing and interpolation methods implemented in (Amaral, 2015).
- A precision model and data structures that handle specific needs.

### 4.3.1 Dependencies

Table 4.4 presents the SPTMesh external dependencies.

| Dependency   | Usage   |
|--------------|---|
| Armadillo    | Linear algebra operations.  |
| BLAS, LAPACK | Armadillo dependencies to provide various matrix decompositions, e.g., SVD. |

---

<sup>14</sup> <https://www.postgresql.org/docs/9.4/static/xfunc-c.html>

<sup>15</sup> See PostgreSQL documentation: Chapter 39 - Procedural Languages

<sup>16</sup> See PostgreSQL documentation: Chapter 35 - Extending SQL.

|      |   |
|------|---|
| GEOS | Manipulation and analysis of 2D geometries in the Cartesian plane and spatial operations. |
|------|---|

Table 4.4 - SPTMesh external dependencies.

### 4.3.2 Data Structures

In this section we discuss the implementation of the main data types in SPTMesh. These include our discrete data model data types.

Table 4.5 presents the SPTMesh data structures used to implement our data model data types.

| Data Model Data Type      | Data Structure Implementation   |
|---------------------------|---|
| TIME                      | We are not yet using dates with locales and time zones and type <i>instant</i> is implemented using the C++ long long data type. This is to be changed in the future. Date <sup>17</sup> , a template-based C++ date and time library, seems to be a good solution to deal with the complexity of dates, locales and time zones.  |
| SPATIAL                   | Type <i>mesh</i> is implemented in class Mesh.<br><br>Type <i>point</i> is implemented in class Point. The GEOS library provides a type <i>point</i> . However, it would be overkill to use the GEOS library to just have a type <i>point</i> available, under certain circumstances. Therefore, SPTMesh implements type <i>point</i> .<br><br>The other SPATIAL types are provided by GEOS.  |
| INTERVAL and PERIOD       | Types: <i>interval</i> and <i>period</i> are implemented in the template classes Interval and Period, respectively. Interval supports the 4 types of intervals but we only use closed-open intervals. A closed-open interval can be used to construct closed-closed, open-closed and open-open intervals. However, no granularity is specified in our model, e.g., to convert a closed-open interval to a closed-closed interval we need a granularity. |
| UNIT, MOVING and FUNCTION | UNIT and MOVING types are implemented in classes: UnitBool, UnitReal, UnitPoint, UnitMesh, MovingBool, MovingReal, MovingPoint and MovingMesh, respectively.<br><br>Type <i>function</i> is implemented in class UnitFunction.  |

Table 4.5 - SPTMesh data structures to implement our discrete data model data types.

Table 4.6 presents other SPTMesh relevant data structures.

<sup>17</sup> <https://howardhinnant.github.io/date/date.html>

| Data Type      | Data Structure Implementation   |
|----------------|---|
| PrecisionModel | SPTMesh implements the same precision model that GEOS uses, in class PrecisionModel. The precision model provides 3 different rounding methods: Fixed precision, Floating double-precision and Floating single-precision. When using the C API SPTMesh will use Floating double-precision. The C++ API is less restrictive and allows its clients to choose the precision model to be used. However, the precision model is something to be looked at more carefully, e.g., we can have: objects created with different precision models or user custom precision models. |
| WKTRReader     | The WKTRReader class constructs UNIT and MOVING types from their Spatiotemporal Well-Known Text (STWKT <sup>18</sup> ) representations and uses a precision model.  |
| Correspondence | The Correspondence class reads the correspondence between 2 polygons, a source and a target polygons, generated by the data acquisition tools implemented in (Mesquita, 2013) and it can further process them.  |

Table 4.6 - Other relevant SPTMesh data structures.

Next we discuss the implementation of the *umesh* type because it is the most important data structure proposed in this work.

### UnitMesh

The *umesh* type implementation, in class UnitMesh, adds 2 new attributes,  $t_b$ ,  $t_e$ , uses a UnitPoint to represent the evolution of the position of the mesh's centroid and it is not decoupled from the interpolation method. The UnitMesh data structure is defined as follows:

$\text{UnitMesh} = \{(I, t_b, t_e, c, P, Q, pStar, rScale, rGamma) \mid I \in \text{Interval}(\text{long long}), t_b, t_e \in \text{long long}, c \in \text{UnitPoint}, P, Q \in \text{Mesh}, pStar, rScale \in \text{mat}, rGamma \in \text{vector<double> such that}$

$$(i) \quad t_b < t_e \wedge (I.t_b \geq t_b \wedge I.t_e \leq t_e)\}$$

(i)  $I$  must be equal or contained by the original interval.

Where:

- $I$  is the interval in which the unit is defined.
- $t_b$  and  $t_e$  represent the begin and end instants of the original interval, respectively.
- $c$  represents the evolution of the position of the mesh's centroid.
- $P$  and  $Q$  represent the original source and target meshes at the begin and end instants of the original interval, respectively.
- $\text{mat}$  is an Armadillo data type that represents a matrix.
- $\text{vector}$  is the C++ standard library vector data type.

<sup>18</sup> STWKT is not a standard. We are introducing this concept to represent a WKT for *unit* and *moving* types.

- pStar, rScale and rGamma are the interpolation components, computed by the interpolation method, used to interpolate the mesh during  $I$ .

The attributes  $t_b$  and  $t_e$  are introduced because of copy operations. The evolution of the mesh's position during  $I$  is stored using a UnitPoint and an important implementation detail is the fact that UnitMesh is not decoupled from the interpolation method. This is discussed in section 4.3.8.

To avoid a possible successive loss of precision, when creating new unit or moving objects from existing ones, we agreed that: given  $u, u^* \in \text{UnitMesh} \wedge I, I^* \in \text{Interval}$ , such that

$u = (I, t_b, t_e, c, P, Q, pStar, rScale, rGamma)$ , then

$\forall u^*, I^*: I^* \subseteq I \wedge \text{copy}(u^*, u) \Rightarrow u^* = (I^*, I.t_b, I.t_e, c, P, Q, pStar, rScale, rGamma)$

Where:

- $\text{copy}(a, b)$  means that  $a$  is a copy of  $b$ .

When creating a new UnitMesh object from an existing one if we use interpolated meshes for the new object source and target meshes we can induce a possible successive loss of precision. Therefore, we copy the source and target meshes, the evolution of the position of the mesh's centroid and the interpolation components to the new object without any changes.

### 4.3.3 Operations

In this subsection we discuss the SPTMesh implementation of the discrete data model operations proposed in section 3.4. We present and discuss only the operations that were implemented.

The notation used in this section follows the convention in Table 4.7.

| Symbol         | Description  |
|----------------|--|
| $t$            | An <i>instant</i> .                                      |
| $P$            | A <i>period(instant)</i> .                               |
| $\mu, \varphi$ | 2 moving types, i.e., $\mu, \varphi \in \text{MOVING}$ . |

Table 4.7 - Notation.

| Operation  | Discrete data model Operation                  | Implementation   |
|------------|--|--|
| equals     | $\varphi \times \varphi \rightarrow mbool$ .   | Implemented for all MOVING types in their respective implementation classes.<br>2 MOVING types are equal iff they have: <ul style="list-style-type: none"> <li>• The same number of units.</li> <li>• Exactly the same units.</li> </ul> |
| intersects | $mmesh \times mmesh \times t \rightarrow bool$ | Implemented in class MovingMesh.   |

Table 4.8 - Predicate operations.

| Operation    | Discrete data model Operation   | Implementation   |
|--------------|---|--|
| intersection | $m\text{mesh} \times m\text{mesh} \times t \rightarrow \text{geometrycollection} \cup \text{polygon}$ | Implemented in class MovingMesh. The operation returns a Well-Known Text (WKT) representation of the intersection. |

Table 4.9 - Set operations.

| Operation | Discrete data model Operation                    | Implementation                   |
|-----------|--|----------------------------------|
| area      | $m\text{mesh} \times t \rightarrow \text{real}$  | Implemented in class MovingMesh. |
|           | $m\text{mesh} \times P \rightarrow m\text{real}$ | Implemented in class MovingMesh. |

Table 4.10 - Numeric operations.

| Operation | Discrete data model Operation | Implementation   |
|-----------|-------------------------------|--|
| deftime   | $\varphi \rightarrow P$       | Implemented for all MOVING types in their respective implementation classes. |

Table 4.11 - Projection of MOVING types to Domain and Range.

| Operation | Discrete data model Operation                      | Implementation   |
|-----------|--|--|
| atinstant | $m\text{mesh} \times t \rightarrow \text{polygon}$ | Class MovingMesh. Implementation does not return a <i>polygon</i> type. It returns a WKT representation of the <i>polygon</i> . This is because we are possibly sending spatial information to an external system. |
|           | $m\text{point} \times t \rightarrow \text{point}$  | Class MovingPoint. Implementation does not return a <i>point</i> type for the same reason as above. In this case 2 options are provided: return a WKT representation of the point or its x and y values.           |
|           | $m\text{real} \times t \rightarrow \text{real}$    | Class MovingReal.  |
|           | $m\text{bool} \times t \rightarrow \text{bool}$    | Class MovingBool.  |
| atperiod  | $\varphi \times P \rightarrow \varphi$             | Implemented for all MOVING types in their respective implementation classes.   |
| present   | $\varphi \times t \rightarrow \text{bool}$         | Implemented for <i>mmesh</i> only in class MovingMesh.   |
|           | $\varphi \times P \rightarrow m\text{bool}$        | Implemented for <i>mmesh</i> only in class MovingMesh.   |

Table 4.12 - Interaction of MOVING types with values in Domain and Range.

| Constructor | Discrete data model Operation     | Implementation  |
|-------------|-----------------------------------|---|
| UnitBool    | $(\dots) \rightarrow \text{UNIT}$ | The constructors are implemented in the classes that implement the respective type. |
| UnitReal    | $\rightarrow \text{MOVING}$       |   |



|  |  |   |
|--|--|---|
| UnitPoint<br>UnitMesh<br>MovingBool<br>MovingReal<br>MovingPoint<br>MovingMesh |  | The WKTRReader class constructs UNIT and MOVING types from their STWKT forms. |
|--|--|---|

Table 4.13 - Constructors.

| Operation | Signature                                  | Implementation  |
|-----------|--|---|
| $\psi$    | List $\times$ List $\rightarrow$ Mesh      | <p>This operation constructs a <i>mesh</i> type.</p> <p>The triangulation method, used to create mesh objects, needs 2 polygons, with a correspondence between them, to work.</p> <p>Moreover, we don't work directly with polygons. Instead, we have a list of their boundary points. But this can change in the future.</p> <p>We use the following generic algorithm to construct 2 <i>mesh</i> types from the lists of boundary points of 2 polygons with a correspondence:</p> <pre> P, Q <math>\in</math> List P*, Q* <math>\in</math> Mesh copyTo(P, P*) copyTo(Q, Q*) triangulate(P*, Q*, triangulationMethod) smoothing(P*, smoothingMethod) (1) smoothing(Q*, smoothingMethod) (2) </pre> <p>(1) and (2) are optional.</p> <p>Where:</p> <ul style="list-style-type: none"> <li>• P, Q are the lists of boundary points of 2 polygons, a source and a target polygons.</li> <li>• copyTo(<i>a</i>, <i>b</i>) means that <i>a</i> is copied to <i>b</i>.</li> <li>• triangulate(<i>a</i>, <i>b</i>, <i>c</i>) means that <i>a</i> and <i>b</i> will be triangulated using method <i>c</i>.</li> <li>• smoothing(<i>a</i>, <i>b</i>) means that <i>a</i> will be smoothed using method <i>b</i>.</li> </ul> |
| toWKT     | SPATIAL $\rightarrow$ string <sup>19</sup> | This operation returns a WKT representation of a  |

<sup>19</sup> Our data model does not define the data type string but we use it in its implementation. Type string is given by the underlying programming language respective type.

|              |  |   |
|--------------|--|---|
|              |  | spatial object. The WKT format is a standard way of exchanging spatial objects with external systems, e.g., GEOS and PostGIS.                                       |
| UnitFunction | (...) → FUNCTION                               | This operation constructs a <i>function</i> type. It is implemented in class UnitFunction.  |
| add          | UNIT → MOVING                                  | This operation adds a unit to a moving object. It is implemented for all MOVING types in their respective implementation classes.                                   |
| toSTWKT      | MOVING → <i>string</i><br>UNIT → <i>string</i> | This operation returns a STWKT representation of UNIT and MOVING types. It is implemented for all UNIT and MOVING types in their respective implementation classes. |

Table 4.14 - Other interesting operations.

#### 4.3.4 Triangulation, Smoothing and Interpolation Methods

In this subsection we discuss the implementation of the triangulation, smoothing and interpolation methods. In section 5.2 we present some results obtained using these methods.

| Method        | Implementation                                  |
|---------------|---|
| Triangulation | Implemented in class CompatibleTriangulation.   |
| Smoothing     | Implemented in class SimpleSmoothing.           |
| Interpolation | Implemented in class OptimalRigidInterpolation. |

Table 4.15 - Triangulation, smoothing and interpolation methods implementation classes.

The triangulation, smoothing and interpolation methods implementation, see Table 4.15, offers the same functionality as in the original implementation in (Amaral, 2015). The implementation of these methods has been optimized:

- Processing is done in memory.
- We simplified some operations, e.g., the computation of the interpolation components translates the triangles directly to the origin, i.e., (0, 0), instead of translating the polygon to the origin and then the triangles.
- The triangulation method uses a lookup table to avoid the re-computation of distances between vertices and uses an algorithm that is different from the original implementation to split a polygon in order to perform that operation in only one pass.

In this implementation the triangulation and smoothing methods use the GEOS library to validate the mesh geometry and to avoid triangle overlaps during the triangulation process. The interpolation method is not verifying geometry validity for interpolated meshes.

We also changed the interpolation method to use the area-weighted average rotation over all the triangles when computing the minimized rotation angles, as proposed in (Baxter et al., 2008). This allows us to have rotations like the one in Figure 4.6.

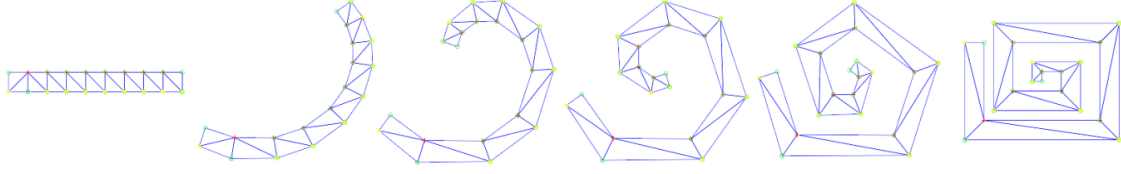


Figure 4.6 - Interpolation of a geometry that coils during an interval.

### 4.3.5 Continuity for Unit Types

SPTMesh defines 2 constant values that are used to help establish continuity for UNIT types, see Table 4.16.

| Value                              | Default Value | Description   |
|------------------------------------|---------------|---|
| RealContinuityEpsilon ( $\xi_p$ )  | 0.00001       | This value is used to help <sup>20</sup> establish continuity for unit point objects. See section 6.4.2 for a discussion on why $\xi_p$ is not used to establish continuity for unit real objects.  |
| MeshContinuityDelta ( $\delta_s$ ) | 0.85          | <p>2 mesh objects are continuous if:</p> <ul style="list-style-type: none"> <li><math>\frac{m_i \cap m_j}{m_i \cup m_j} \leq \delta_s</math></li> <li><math>distance(m_i, m_j) \leq \xi_p, m_i, m_j \in \text{Mesh},</math></li> </ul> <p><math>distance(m_i, m_j)</math> refers to the distance between the mesh centroids. With this definition we can define continuity for unit mesh objects.</p> <p>See section 6.4.1 for a discussion on why <math>distance(m_i, m_j)</math> is not actually used to establish continuity for mesh objects.</p> |

Table 4.16 - The SPTMesh values used to help establish a notion of continuity for UNIT types.

Both  $\xi_p$  And  $\delta_s$  were empirically set during the tests phase using real data. However, given the limited size of our dataset the constants and formulas used to establish continuity need further evaluation and might be adapted for larger datasets.

### 4.3.6 A Spatiotemporal Well-Known Text Form for Moving Objects

In this section we use the notation as given in Table 4.17.

| Symbol | Description |
|--------|-------------|
|--------|-------------|

<sup>20</sup> The unit's interval is also used to establish continuity for unit objects.

|  |   |
|--|---|
| $t_b, t_e \in \text{instant}$                            | Begin and end instants of an <i>interval</i> .  |
| $v \in \text{bool}$                                      | A value in {true, false} or equivalently in {1, 0}.   |
| $v_b, v_e \in \text{double}$                             | Begin and end values, respectively.   |
| $\text{type}, \text{typeX}, \text{typeY} \in \text{int}$ | Type of interpolation function and type of interpolation function for the x and y components, respectively. |
| $x_b, y_b, x_e, y_e \in \text{double}$                   | Begin and end x and y components, respectively.   |

Table 4.17 - Notation.

The STWKT was defined in order to express UNIT and MOVING types in a common format. The STWKT form is not a standard. This concept ‘extends’ the WKT standard form for expressing spatial objects given by the OpenGIS specification. We defined the STWKT form presented in Table 4.18.

| Type          | STWKT Format  |
|---------------|---|
| <i>ubool</i>  | UNITBOOL( $t_b t_e v$ )<br>E.g.: UNITBOOL(1491214210 1491300610 1)  |
| <i>ureal</i>  | UNITREAL( $t_b t_e v_b v_e \text{type}$ )<br>E.g.: UNITREAL(1491214210 1491300610 7.1 9.7 1)  |
| <i>upoint</i> | UNITPOINT( $t_b t_e x_b y_b x_e y_e \text{typeX typeY}$ )<br>E.g.: UNITPOINT(1491214210 1491300610 3.1 4.5 4 7.8 1 1)   |
| <i>umesh</i>  | UNITMESH( $t_b t_e, (\text{source\_polygon\_coordinates}), (\text{target\_polygon\_coordinates})$ )<br>UNITMESH( $t_b t_e, (x_1 y_1, \dots, x_n y_n), (x_1 y_1, \dots, x_n y_n)$ )<br>E.g.: UNITMESH(1491214210 1491300610, (1052 987, ..., 1096 1095),<br>(1055 999, ..., 1104 1074))<br>( $x_1 y_1, \dots, x_n y_n$ ) is not closed, i.e., $x_1 = x_n \wedge y_1 = y_n$ . |
| <i>mbool</i>  | MOVINGBOOL(UNITBOOL <sub>1</sub> , ..., UNITBOOL <sub>n</sub> )<br>MOVINGBOOL( $t_{b1} t_{e1} v_1, \dots, t_{bn} t_{en} v_n$ )<br>E.g.:<br>MOVINGBOOL EMPTY<br>MOVINGBOOL(1491041410 1491127810 1, ..., 1491214210 1491300610 0)  |
| <i>mreal</i>  | MOVINGREAL(UNITREAL <sub>1</sub> , ..., UNITREAL <sub>n</sub> )<br>MOVINGREAL( $(t_b t_e v_b v_e \text{type})_1, \dots, (t_b t_e v_b v_e \text{type})_n$ )<br>E.g.:<br>MOVINGREAL EMPTY<br>MOVINGREAL( $(1491041410 1491127810 1 5 1), \dots, (1491214210 1491300610 -7 2 1)$ )   |
| <i>mpoint</i> | MOVINGPOINT(UNITPOINT <sub>1</sub> , ..., UNITPOINT <sub>n</sub> )<br>MOVINGPOINT( $((t_b t_e x_b y_b x_e y_e \text{typeX typeY})_1, \dots, (t_b t_e x_b y_b x_e y_e \text{typeX typeY})_n)$ )<br>E.g.:<br>MOVINGPOINT EMPTY<br>MOVINGPOINT(1491127810 1491214210 2 2 3 4.5 1 1), ..., (1491214210 1491300610 3 4.5   |

|              |   |
|--------------|---|
|              | 4 7.8 1 1)  |
| <i>mmesh</i> | MOVINGMESH(UNITMESH <sub>1</sub> , ..., UNITMESH <sub>n</sub> )<br>MOVINGMESH((t <sub>b</sub> t <sub>e</sub> , (x <sub>1</sub> y <sub>1</sub> , ..., x <sub>n</sub> y <sub>n</sub> ), (x <sub>1</sub> y <sub>1</sub> , ..., x <sub>n</sub> y <sub>n</sub> )) <sub>1</sub> , ..., (t <sub>b</sub> t <sub>e</sub> , (x <sub>1</sub> y <sub>1</sub> , ..., x <sub>n</sub> y <sub>n</sub> ), (x <sub>1</sub> y <sub>1</sub> , ..., x <sub>n</sub> y <sub>n</sub> )) <sub>n</sub> )<br>E.g.:<br>MOVINGMESH EMPTY<br>MOVINGMESH((1491041410 1491127810, (0 0, ..., 6 0), (6 8, ..., 0 8)), ...) |

Table 4.18 - STWKT formats for UNIT and MOVING types.

### 4.3.7 Architecture

In this subsection we discuss the SPTMesh and GEOS architectures.

The SPTMesh architecture follows the GEOS library architecture. There are naturally differences. E.g., GEOS:

- Has no external dependencies.
- Has a close system of operations.
- Is a complex, mature, well-established project.

Figure 4.7 shows the SPTMesh architecture and available APIs.

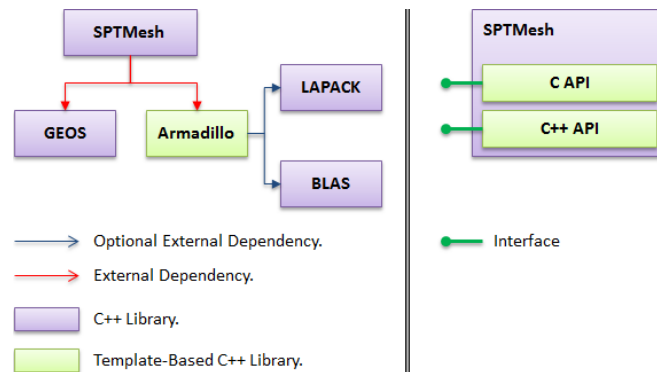


Figure 4.7 - SPTMesh architecture and available APIs.

Figure 4.8 shows the GEOS type system.

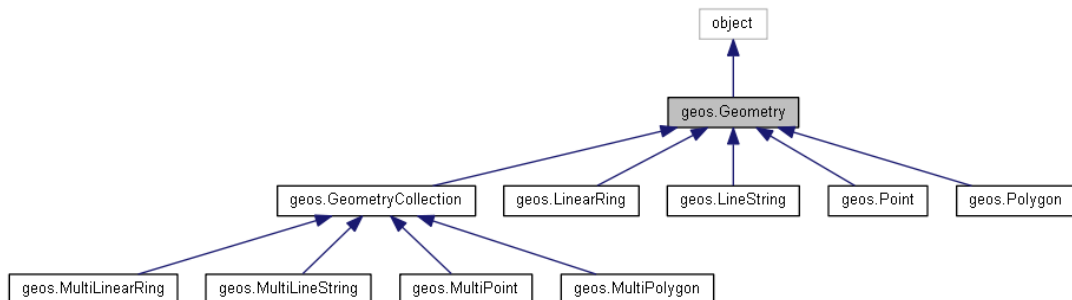


Figure 4.8 - The GEOS type system. Source: GEOS documentation.

Figure 4.9 shows the SPTMesh type system.

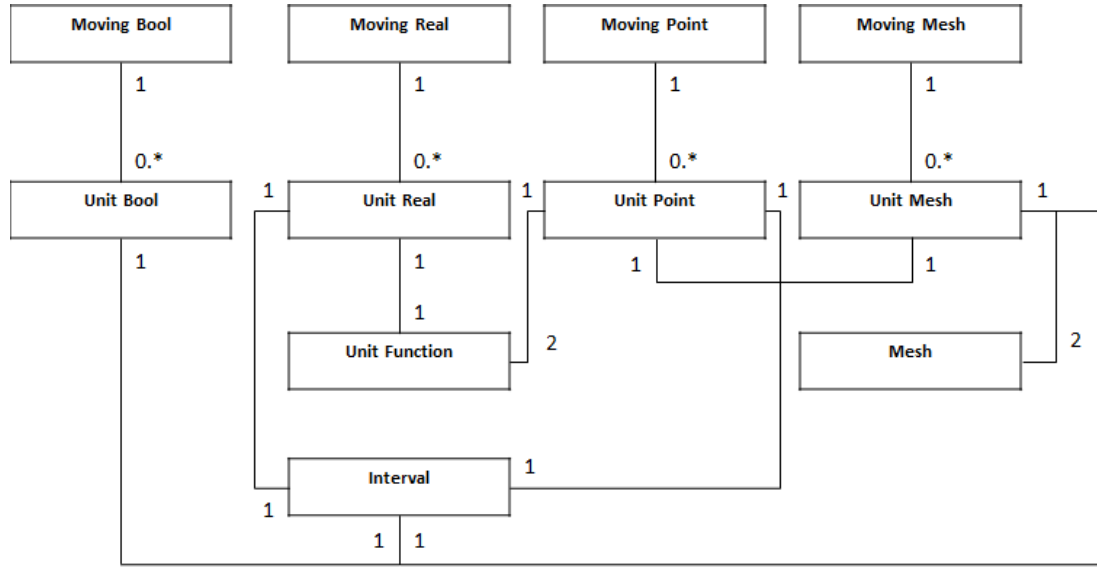


Figure 4.9 - The SPTMesh type system.

#### 4.3.8 Issues

We think that SPTMesh has some architectural limitations.

Type *umesh* implementation, i.e., the UnitMesh class, is not decoupled from the interpolation method. Since it would be interesting to use different interpolation methods the SPTMesh architecture should be modified in the future to allow the framework to be extended with new interpolation methods in a simple and natural way.

The infrastructure needed to decouple UnitMesh from the interpolation method is already created and tested. However, SPTMesh can have clients that store the interpolation components, e.g., MeshGIS, and require them to be constructed at a later time. Therefore, we need a representation that decouples the client from interpolation details. And this is the main reason why UnitMesh is not decoupled from the interpolation method.

The next definition gives us the representation we need.

Given  $IC$ , the set of all admissible interpolation components,  $\lambda$ , an un-ordered list of real numbers,  $\phi$ , a function with an inverse function  $\phi^{-1}$ , assuming:

$\forall \tau \in IC, \exists \phi : \phi(\tau) = \lambda$ , such that

- (i)  $\forall \tau_i, \tau_j \in IC : \tau_i \neq \tau_j \Rightarrow \phi(\tau_i) \neq \phi(\tau_j)$
- (ii)  $\phi(\tau_i) = \phi(\tau_j) \Rightarrow \tau_i = \tau_j$

Where:

- $\tau$  represents the interpolation components of some interpolation method.

That is, we assume that the interpolation components of any admissible interpolation method, independently of what they are and the data structures they use, can be represented by an un-ordered list of real numbers and this representation is unique.

Then, we only need to provide such  $\phi$  and  $\phi^{-1}$  functions for every interpolation method and the SPTMesh clients can use them to store and construct the interpolation components as needed, without worrying about interpolation method details. This is the piece missing in this implementation.

If we implement  $\tau$  using a class  $C$ ,  $\text{UnitMesh} = (I, t_b, t_e, c, P, Q, C)$ , we decouple UnitMesh from the interpolation method. Each interpolation method implements its own  $\phi$  and  $\phi^{-1}$  functions.

Figure 4.10 shows the infrastructure implemented in SPTMesh to decoupled the UnitMesh class from interpolation method details.

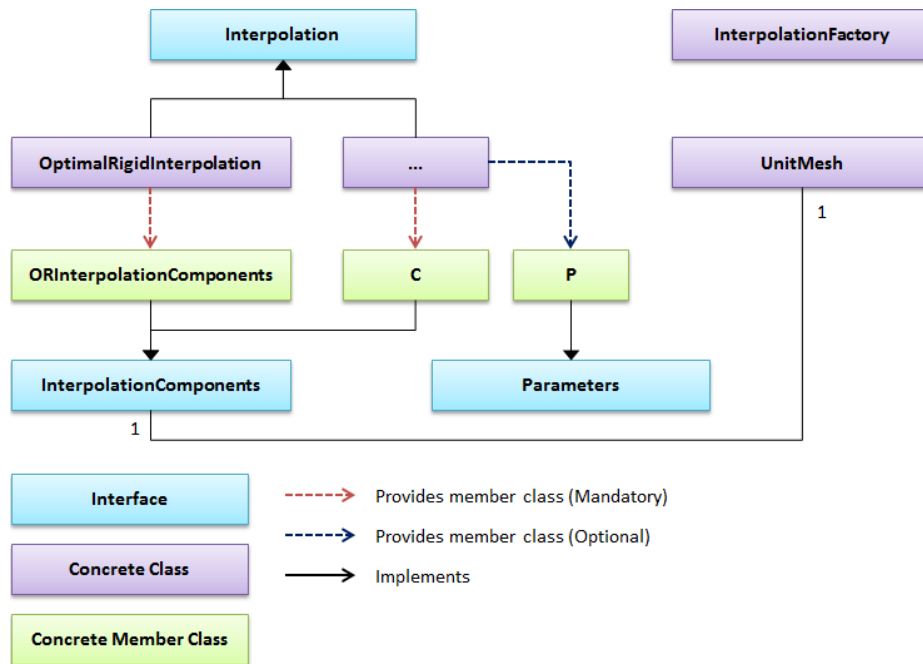


Figure 4.10 - SPTMesh interpolation method implementation classes and decoupling infrastructure.

Table 4.19 presents a description for the main classes shown in Figure 4.10.

| Class                   | Main Responsibilities  |
|-------------------------|--|
| InterpolationComponents | Holds the interpolation components of an interpolation method and provides an interpolated value at a specified instant in $[0, 1]$ , using the concrete class implementation of the respective interpolation method, i.e., when UnitMesh needs an interpolated value it will use the InterpolationComponents class that it holds to get it. |
| Parameters              | Used to define the parameters' values of an interpolation method.  |

Table 4.19 - Main classes and responsibilities.

Other issues include:

- The current implementation does not provide a *mobject* type that, potentially, can make the API more elegant and enable us to have a closed system of operations. A *mobject* type is an abstract type that can represent any MOVING type.
- The current implementation does not provide a *uobject* type that can represent any UNIT type.
- The precision model is well-defined but it is not well-integrated in the SPTMesh architecture. It would be interesting to have a moving object factory to handle the construction of moving objects. And such a factory would own and use the precision model. This follows the architecture of the GEOS library.
- The architecture does not allow user-defined:
  - Unit functions and this could be particularly interesting for *ureal* and *upoint* types.
  - Precision models.

Figure 4.11 shows a type system with a *mobject* and *uobject* types.

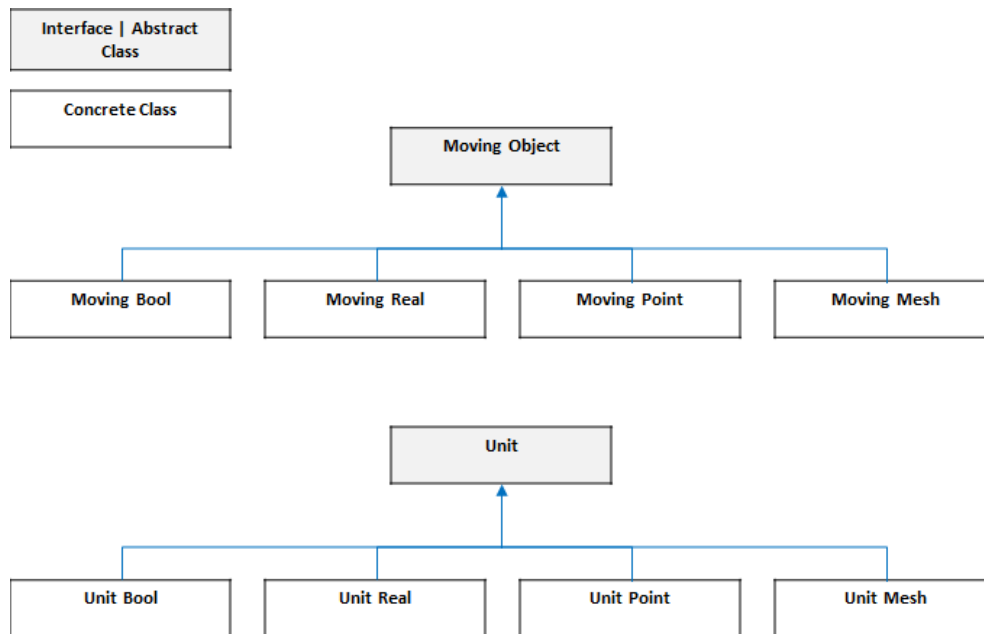


Figure 4.11 - A type system with a *mobject* and *uobject* types.

- We are not working with dates to define the unit's interval. We want type *instant* to represent a date with a specific locale and time zone.
- MOVING and UNIT types should be associated with a precision model.

### 4.3.9 Usage and Installation

To use SPTMesh with MeshGIS just copy SPTMesh and its external dependencies to the bin directory of the PostgreSQL installation that is being using. E.g., in a windows operating system with PostgreSQL 9.4:

- Copy the following files to C:\Program Files\PostgreSQL\9.4\bin:
  - lapack\_win32\_MT.dll



- blas\_win32\_MT.dll
- geos\_c.dll
- sptmesh.dll
- sptmesh\_c.dll

Notes:

- Assuming that PostgreSQL is installed in C:\Program Files\PostgreSQL\9.4.
- In windows systems the geos\_c.dll can be found in the OSGeo4W distribution for windows.
- We obtained lapack\_win32\_MT.dll and blas\_win32\_MT.dll from<sup>21</sup>.

To use SPTMesh with another application: put SPTMesh and its external dependencies in the same directory with the executable or in some other directory where the application can find them.

If using the C API the client must call SPTMESH\_init\_r before using SPTMesh and SPTMESH\_finish\_r when done. This is mandatory. An example of usage follows:

```
ContextHandle handle = SPTMESH_init_r(ST_callback_function);
// Do work.
SPTMESH_finish_r(handle);
...
```

Finally, SPTMesh objects take ownership of other objects. Read the SPTMesh documentation or the source code .h files before using the functions provided to destroy SPTMesh objects, to avoid segmentation errors when releasing memory owned by SPTMesh objects. This mechanism follows the GEOS architecture.

## 4.4 MeshGIS - A Spatiotemporal Extension for PostgreSQL

MeshGIS is a C library that follows the architecture of PostGIS:

- It is a spatiotemporal database extension<sup>22</sup> for PostgreSQL.
- Uses SPTMesh to analyze and manipulate moving objects.
- Allows the moving objects provided by SPTMesh to be stored on PostgreSQL and manipulated using SQL.

Table 4.20 shows the MeshGIS external dependencies.

| Dependency | Usage   |
|------------|---|
| PostgreSQL | MeshGIS is completely dependent on PostgreSQL. That means that MeshGIS is only guaranteed to work with the PostgreSQL version with which it was compiled. |
| SPTMesh    | Manipulation and analysis of moving objects.  |

Table 4.20 - MeshGIS external dependencies.

<sup>21</sup> <http://ylzhao.blogspot.pt/2013/10/blas-lapack-precompiled-binaries-for.html>

<sup>22</sup> Not a complete spatiotemporal database extension.

#### 4.4.1 Data Structures

MeshGIS implementation and data structures are based on PostGIS. Of particular interest in PostGIS is the way it uses the GEOS library, how it exchanges data with PostgreSQL and the data structures it defines to represent geometries. The PostGIS data structures used to represent geometries are defined as C typedef structs and they include<sup>23</sup>: POINTARRAY, GSERIALIZED, LWGEOM, LWPOINT, LWLINE, LWPOLY, LWMPOINT, LWMPOLY and LWCOLLECTION, among others. POINTARRAY, GSERIALIZED and LWGEOM are particularly interesting, see Table 4.21. The others represent specific geometry types or collections of geometries.

| Data Structure | Description  |
|----------------|--|
| GSERIALIZED    | The data structure used to exchange data with PostgreSQL, i.e., the data structure used to send data to and to receive data from PostgreSQL. A PostgreSQL data type for variable size user-defined data types. |
| LWGEOM         | Represents an abstract type that can represent any geometry type. This is useful and simplifies the API.   |
| POINTARRAY     | Represents an array of points. The data structures: LWPOINT, LWLINE, LWTRIANGLE, LWCIRCSTRING and LWPOLY, which are the building blocks of other data structures, e.g., collections, use POINTARRAY.           |

Table 4.21 - PostGIS data structures.

Also interesting for our discussion, is the connection between PostgreSQL, PostGIS and the GEOS library. We have 2 main use cases:

- a) We have a geometry stored on PostgreSQL that we want to manipulate or analyze, i.e., we need to use the GEOS library.
- b) We have a geometry that we want to store on PostgreSQL. It can be a geometry obtained as an intermediate result or from a standard representation.

In the first case:

1. PostGIS receives a GSERIALIZED data structure from PostgreSQL. Technically it is a data structure with the same structure as GSERIALIZED.
2. That data structure is transformed into a PostGIS data structure for the respective geometry type, e.g., LWPOLY.
3. The new data structure will pass through the system as a LWGEOM.
4. At the end of the chain, the new data structure is transformed into a GEOS data structure, using the GEOS C API.
5. The GEOS C API is used to manipulate the geometry.

---

<sup>23</sup> See: [http://postgis.net/docs/doxygen/2.2/da/de7/liblwgeom\\_8h.html](http://postgis.net/docs/doxygen/2.2/da/de7/liblwgeom_8h.html) or the liblwgeom.h file in the PostGIS source code, for a complete list.

In the second case, we have a GEOS data structure or PostGIS used the GEOS C API to get one. So we start with a GEOS data structure:

1. The GEOS data structure is transformed into a PostGIS data structure for the respective geometry type, e.g., LWPOLY.
2. The new data structure will pass through the system as a LWGEOM.
3. At the end of the chain, the new data structure is transformed into a GSERIALIZED data structure and sent to PostgreSQL for storage.

See POSTGIS2GEOS and GEOS2POSTGIS in lwgeom\_geos.c, in the PostGIS source code, for more details.

MeshGIS data structures to represent moving objects are also defined as C typedef structs and they are presented in Table 4.22.

| Data Structure             | Description  |
|----------------------------|--|
| ArrayOfX                   | A generic array to hold units or other elements, e.g., UnitReal, UnitBool, UnitPoint and Matrix2x2.                      |
| UnitFunction               | A <i>function</i> type.  |
| UnitInterval               | An <i>interval</i> type.   |
| UnitBool                   | An <i>ubool</i> type.  |
| UnitReal                   | A <i>ureal</i> type.   |
| UnitPoint                  | An <i>upoint</i> type.   |
| UnitMesh                   | An <i>umesh</i> type.  |
| SerializedPostgreSQLObject | This is the GSERIALIZED data structure with a different name. It has the same purpose as the GSERIALIZED data structure. |
| SerializedMovingObject     | A generic MOVING type.   |
| SerializedMovingX          | Represents <i>mbool</i> , <i>mreal</i> and <i>mpoint</i> types.  |
| SerializedMovingMesh       | A <i>mmesh</i> type.   |

Table 4.22 - MeshGIS data structures to represent moving objects.

MeshGIS also defines data structures to represent the interpolation components, see Table 4.23.

| Data Structure | Description  |
|----------------|--|
| Matrix2x2      | A 2x2 matrix representation. Used to hold the scale matrix of a triangle.          |
| Matrix2x3      | A 2x3 matrix representation. Used to hold the transformation matrix of a triangle. |
| Triangles      | The mesh's triangles.  |

Table 4.23 - MeshGIS data structures to represent the interpolation components.

MeshGIS follows the PostGIS paradigm. It provides:

- A generic array data structure that follows the POINTARRAY definition.

- A GSERIALIZED data structure.
- A data structure that represents a generic MOVING type.
- Data structures that represent specific MOVING types. Although it uses SerializedMovingX to represent *mbool*, *mreal* and *mpoint* types.

It also defines data structures to represent interpolation components but this is, we think, a weakness and it should be changed in the future. We want to decouple MeshGIS from interpolation details.

MeshGIS data structures relationships are somewhat complex to represent in a diagram, e.g., some data structures are constructed using arrays of other data structures. A possible simplified diagram of these relationships is shown in Figure 4.12.

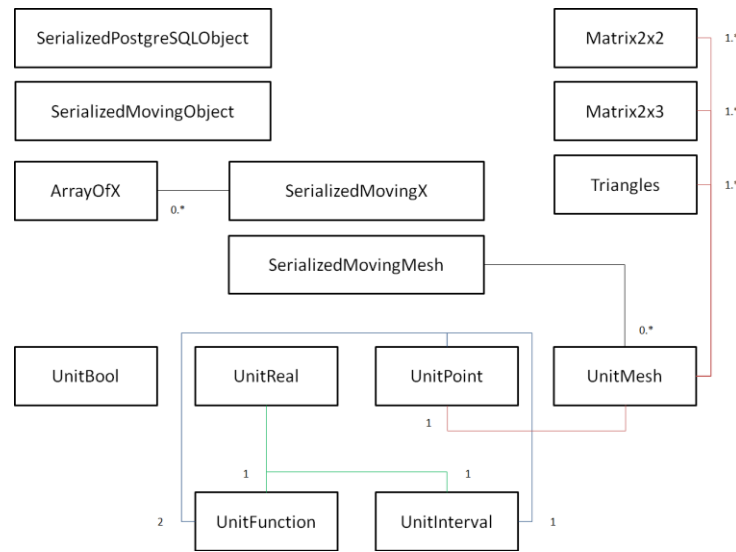


Figure 4.12 - MeshGIS data structures relationships simplified diagram overview.

MeshGIS, PostgreSQL and the GEOS library connect in the same way that PostGIS, PostgreSQL and the GEOS library connect.

UnitMesh is a data structure relatively more complex than the others and it has a variable size. As a consequence, MovingMesh objects are treated as a special case, e.g., the other moving objects are passed through the system as a SerializedMovingObject, i.e., a generic MOVING type, while MovingMesh objects will use the SerializedMovingMesh data structure. This should change in the future.

If we have a moving object stored on PostgreSQL that we want to manipulate or analyze, then:

1. MeshGIS receives a SerializedPostgreSQLObject data structure from PostgreSQL.
2. That data structure is transformed into a MeshGIS data structure for the respective MOVING type, e.g., SerializedMovingX.
3. The new data structure will pass through the system as a SerializedMovingObject or a SerializedMovingMesh.

4. At the end of the chain, the new data structure is transformed into a SPTMesh data structure, using its C API.
5. The SPTMesh C API is used to manipulate the moving object.

If we have a SPTMesh data structure that we want to store in PostgreSQL.

1. The SPTMesh data structure is transformed into a MeshGIS data structure for the respective MOVING type.
2. The new data structure will pass through the system as before.
3. At the end of the chain, the new data structure is transformed into a SerializedPostgreSQLObject data structure and sent to PostgreSQL for storage.

SPTMesh does not provide a generic MOVING type. As a consequence, MeshGIS does not have functions that map directly to the POSTGIS2GEOS and GEOS2POSTGIS PostGIS functions.

#### 4.4.2 Operations

Table 4.24, Table 4.25, Table 4.26 and Table 4.27 present the MeshGIS operations on MOVING types. The respective binding SQL operations are also given.

| MeshGIS Function      | SQL Binding Operation                          | Description   |
|-----------------------|--|---|
| MovingBool_in         | a) MovingBool_in<br>b) ST_MovingBool_FromSTWKT | a) Mandatory by the PostgreSQL rules for user-defined types.<br>b) Constructs a MovingBool from its STWKT form. |
| MovingBool_out        | MovingBool_out                                 | Mandatory by the PostgreSQL rules for user-defined types.   |
| MovingBool_create     | ST_MovingBool_CreateEmpty                      | Creates an empty MovingBool.  |
| MovingBool_add        | ST_Add_UnitBool                                | Adds a UnitBool to a MovingBool.  |
| MovingBool_del        | ST_Del_UnitBool                                | Deletes a UnitBool from a MovingBool.   |
| MovingBool_size       | ST_Get_Size                                    | Gets the number of units of a MovingBool.   |
| MovingBool_deftime    | ST_MovingBool_Get_DefTime                      | Gets the period in which a MovingBool is defined.   |
| MovingBool_atperiod   | ST_MovingBool_Get_AtPeriod                     | Gets a MovingBool that is defined: during a given period over a given MovingBool.                               |
| MovingBool_equals     | Equals   | Checks if 2 MovingBool objects are equal.   |
| MovingBool_atinstant2 | ST_MovingBool_Get_AtInstant2                   | Gets the MovingBool object at an instant.   |

Table 4.24 - MeshGIS and its binding SQL operations for SPTMesh MovingBool objects.

| MeshGIS Function | SQL Binding Operation | Description                              |
|------------------|-----------------------|--|
| MovingReal_in    | a) MovingReal_in      | c) Mandatory by the PostgreSQL rules for |

|                      |                             |   |
|----------------------|-----------------------------|---|
|                      | b) ST_MovingReal_FromSTWKT  | user-defined types.<br>d) Constructs a MovingReal from its STWKT.                 |
| MovingReal_out       | MovingReal_out              | Mandatory by the PostgreSQL rules for user-defined types.                         |
| MovingReal_create    | ST_MovingReal_CreateEmpty   | Creates an empty MovingReal.  |
| MovingReal_add       | ST_Add_UnitReal             | Adds a UnitReal to a MovingReal.  |
| MovingReal_del       | ST_Del_UnitReal             | Deletes a UnitReal from a MovingReal.   |
| MovingReal_size      | ST_Get_Size                 | Gets the number of units of a MovingReal.   |
| MovingReal_deftime   | ST_MovingReal_Get_DefTime   | Gets the period in which a MovingReal is defined.                                 |
| MovingReal_atperiod  | ST_MovingReal_Get_AtPeriod  | Gets a MovingReal that is defined: during a given period over a given MovingReal. |
| MovingReal_atinstant | ST_MovingReal_Get_AtInstant | Gets the MovingReal object at an instant.   |

Table 4.25 - MeshGIS and its binding SQL operations for SPTMesh MovingReal objects.

| MeshGIS               | SQL Binding Operation                            | Description   |
|-----------------------|--|---|
| MovingPoint_in        | a) MovingPoint_in<br>b) ST_MovingPoint_FromSTWKT | e) Mandatory by the PostgreSQL rules for user-defined types.<br>f) Constructs a MovingPoint from its STWKT. |
| MovingPoint_out       | MovingPoint_out                                  | Mandatory by the PostgreSQL rules for user-defined types.   |
| MovingPoint_create    | ST_MovingPoint_CreateEmpty                       | Creates an empty MovingPoint.   |
| MovingPoint_add       | ST_Add_UnitPoint                                 | Adds a UnitPoint to a MovingPoint.  |
| MovingPoint_del       | ST_Del_UnitPoint                                 | Deletes a UnitPoint from a MovingPoint.   |
| MovingPoint_size      | ST_Get_Size                                      | Gets the number of units of a MovingPoint.  |
| MovingPoint_deftime   | ST_MovingPoint_Get_DefTime                       | Gets the period in which a MovingPoint is defined.  |
| MovingPoint_atperiod  | ST_MovingPoint_Get_AtPeriod                      | Gets a MovingPoint that is defined: during a given period over a given MovingPoint.                         |
| MovingPoint_atinstant | ST_MovingPoint_Get_AtInstant                     | Gets the MovingPoint object at an instant.  |

Table 4.26 - MeshGIS and its binding SQL operations for SPTMesh MovingPoint objects.

| MeshGIS       | SQL Binding Operation | Description                    |
|---------------|-----------------------|--------------------------------|
| MovingMesh_in | a) MovingMesh_in      | g) Mandatory by the PostgreSQL |

|                             |                            |   |
|-----------------------------|----------------------------|---|
|                             | b) ST_MovingMesh_FromSTWKT | rules for user-defined types.<br>h) Constructs a MovingMesh from its STWKT.       |
| MovingMesh_out              | MovingMesh_out             | Mandatory by the PostgreSQL rules for user-defined types.                         |
| MovingMesh_create           | ST_MovingMesh_CreateEmpty  | Creates an empty MovingMesh.  |
| MovingMesh_add              | ST_Add_UnitMesh            | Adds a UnitMesh to a MovingMesh.  |
| MovingMesh_del              | ST_Del_UnitMesh            | Deletes a UnitMesh from a MovingMesh.   |
| MovingMesh_size             | ST_Get_Size                | Gets the number of units of a MovingMesh.   |
| MovingMesh_deftime          | ST_Get_DefTime             | Gets the period in which a MovingMesh is defined.                                 |
| MovingMesh_atperiod         | ST_Get_AtInstant           | Gets a MovingMesh that is defined: during a given period over a given MovingMesh. |
| MovingMesh_atinstant        | ST_Get_AtInstant           | Gets the MovingMesh object at an instant.   |
| MovingMesh_atinstant_octave | ST_Get_AtInstant_ToOctave  | Used to see geometries in Octave.   |
| MovingMesh_area_atinstant   | ST_Get_Area                | Gets the MovingMesh object area at an instant.                                    |
| MovingMesh_area_atperiod    | ST_Get_Area_AtPeriod       | Gets the MovingMesh object area at a period.                                      |
| MovingMesh_intersects       | ST_Intersect               | Checks if 2 MovingMesh objects intersect at an instant.                           |
| MovingMesh_intersection     | ST_Intersection            | Gets the intersection of 2 MovingMesh objects at an instant.                      |
| MovingMesh_present          | ST_Present                 | Checks if a MovingMesh object exists at an instant.                               |
| MovingMesh_preent_atperiod  | ST_Get_Present_AtPeriod    | Checks if a MovingMesh object exists at a period.                                 |

Table 4.27 - MeshGIS and its binding SQL operations for SPTMesh MovingMesh objects.

#### 4.4.3 Architecture

Figure 4.13 shows the MeshGIS architecture.

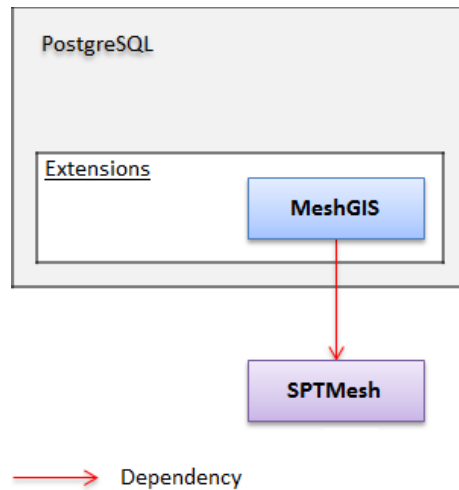


Figure 4.13 - MeshGIS high level architecture overview.

The MeshGIS architecture has as a reference the PostGIS architecture. Figure 4.14 shows a PostGIS high level architecture overview and Table 4.28 presents its external dependencies.

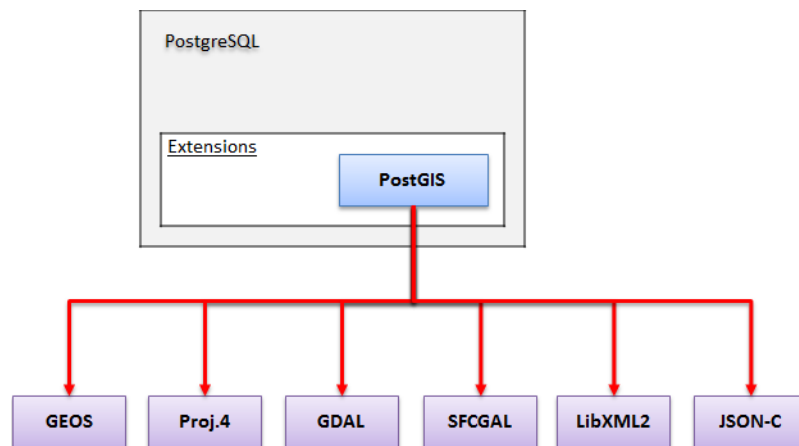


Figure 4.14 - PostGIS high level architecture overview with its external dependencies.

| Dependency | Usage   |
|------------|---|
| GEOS       | Geometry processing.  |
| Proj.4     | Coordinate re-projection functions.                           |
| GDAL       | Raster processing and format support.                         |
| SFCGAL     | Extended 3D support and additional geo-processing algorithms. |
| LibXML2    | XML parsing.  |
| JSON-C     | JSON parsing.   |

Table 4.28 - PostGIS external dependencies.

#### 4.4.4 Usage and Installation

To use MeshGIS:



- Install SPTMesh and its dependencies as described in section 4.3.9.
- Copy meshgis.dll to the PostgreSQL installation lib directory.
- Create the types and functions defined in the Definition.sql files that can be found in the sql directory, in the database that is to be used to store and work with moving objects. See also section 4.5.1.

**Important:**

MeshGIS is only guaranteed to work with the same PostgreSQL version with which it was compiled. To work with a different PostgreSQL version users should change the project settings to point to the intended PostgreSQL installation, a normal PostgreSQL installation should be sufficient, and recompile MeshGIS.

Figure 4.15 shows the MOVING types available in PostgreSQL after installing MeshGIS.

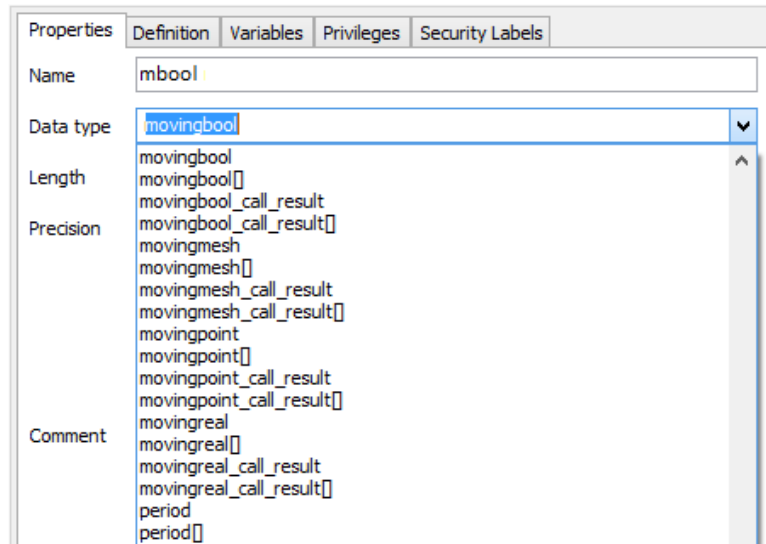


Figure 4.15 - *Moving* types available in PostgreSQL after installing MeshGIS.

## 4.5 A Framework for Future Work and Investigation

We do think that one of the most important contributions of this dissertation is the fact that the implemented components can be used as a framework for future work and investigation in this area. Although there was not enough time to build a ‘plug and play’ kind of architecture it is possible to change the triangulation and smoothing methods in a relatively simple and easy way however this possibility was not completely implemented for the interpolation method.

### 4.5.1 Code Structure

SPTMesh classes are organized into namespaces, see Table 4.29.

| Namespace      | Description  |
|----------------|--|
| correspondence | Operations related to the correspondence between polygons. |
| interpolation  | Interpolation methods.                                     |

|               |   |
|---------------|---|
| io            | Input/output operations.                |
| precision     | Precision model.                        |
| range         | Range types, interval and period types. |
| smoothing     | Smoothing methods.                      |
| spatial       | Spatial types.                          |
| temporal      | Moving and unit types.                  |
| time          | Time types, e.g., instant.              |
| triangulation | Triangulation methods.                  |
| util          | Utility operations.                     |

Table 4.29 - SPTMesh namespaces.

There was a significant effort to document the classes and their functions in our source code and we provide doxygen files to generate the documentation of both SPTMesh and MeshGIS.

Table 4.30 presents the SPTMesh project structure.

| Directory  | Description   |           |             |         |                                |           |                              |           |                                      |            |                                    |
|------------|---|-----------|-------------|---------|--------------------------------|-----------|------------------------------|-----------|--------------------------------------|------------|------------------------------------|
| build      | Visual studio projects for SPTMesh. <table border="1"> <tr> <th>Directory</th><th>Description</th></tr> <tr> <td>sptmesh</td><td>C++ API visual studio project.</td></tr> <tr> <td>sptmesh_c</td><td>C API visual studio project.</td></tr> <tr> <td>tests\c++</td><td>C++ API tests visual studio project.</td></tr> <tr> <td>tests\capi</td><td>C API tests visual studio project.</td></tr> </table> | Directory | Description | sptmesh | C++ API visual studio project. | sptmesh_c | C API visual studio project. | tests\c++ | C++ API tests visual studio project. | tests\capi | C API tests visual studio project. |
| Directory  | Description   |           |             |         |                                |           |                              |           |                                      |            |                                    |
| sptmesh    | C++ API visual studio project.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| sptmesh_c  | C API visual studio project.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| tests\c++  | C++ API tests visual studio project.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| tests\capi | C API tests visual studio project.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| capi       | C API source code.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| doc        | Code documentation and doxygen file to generate the documentation.  |           |             |         |                                |           |                              |           |                                      |            |                                    |
| include    | Armadillo and SPTMesh header files.   |           |             |         |                                |           |                              |           |                                      |            |                                    |
| src        | SPTMesh source files.   |           |             |         |                                |           |                              |           |                                      |            |                                    |
| tests      | Tests source files. <table border="1"> <tr> <th>Directory</th><th>Description</th></tr> <tr> <td>c++api</td><td>C++ API tests source files.</td></tr> <tr> <td>capi</td><td>C API tests source files.</td></tr> </table> <p>We did not use a Tests Framework to perform the tests.</p>  | Directory | Description | c++api  | C++ API tests source files.    | capi      | C API tests source files.    |           |                                      |            |                                    |
| Directory  | Description   |           |             |         |                                |           |                              |           |                                      |            |                                    |
| c++api     | C++ API tests source files.   |           |             |         |                                |           |                              |           |                                      |            |                                    |
| capi       | C API tests source files.   |           |             |         |                                |           |                              |           |                                      |            |                                    |

Table 4.30 - SPTMesh project structure.

Note:

- The SPTMesh visual studio project is linking to the geos\_c.lib distributed with the OSGeo4W<sup>24</sup> distribution for windows.

Table 4.31 presents the MeshGIS project structure.

| Directory | Description  |
|-----------|--|
| meshgis   | Source and header files.   |
| doc       | Code documentation and doxygen file to generate the documentation.                                     |
| sql       | Files with the definition of the SQL types and functions that bind to the MeshGIS types and functions. |

Table 4.31 - MeshGIS project structure.

The visual studio solution for MeshGIS can be found in the root directory.

## 4.5.2 Extending the Framework

To add a new smoothing method to SPTMesh we need to follow the steps presented in Table 4.32.

| Step   | Example and Description  |
|--|--|
| Create a new class that implements the Smoothing interface.                                    | E.g.:<br>LinearSmoothing : public virtual Smoothing<br>This is the class that implements the new method.   |
| The new class must have a member class that implements the Parameters interface.               | E.g.:<br>LinearSmoothingParameters : public virtual Parameters<br>This class is used to pass parameters to the method, i.e., every method can have its own parameters.           |
| Add a new entry to the SmoothingMethod enum defined in the Smoothing class for the new method. | Enum SmoothingMethod<br>{<br>...<br>LinearSmoothingMethod<br>};  |
| Update the SmoothingFactory createInstance function to be able to construct the new method.    | Smoothing* SmoothingFactory::createInstance(...)<br>{<br>...<br>switch (smoothingMethod)<br>{<br>...<br>case LinearSmoothingMethod:<br>return new LinearSmoothing();<br>...<br>} |

<sup>24</sup> We used OSGeo4W version 2.579 with GEOS version 3.5.0-1.

|  |                              |
|--|------------------------------|
|  | <pre>         }     } </pre> |
|--|------------------------------|

Table 4.32 - Adding a new smoothing method to SPTMesh.

To add a new triangulation method to SPTMesh we follow the same steps as above. See the implementation of the smoothing and triangulation methods for more details.

## 4.6 Summary

We implemented 2 components: SPTMesh and MeshGIS. SPTMesh is a framework for moving objects that implements the discrete model presented and proposed in chapter 3. MeshGIS is a spatiotemporal extension for PostgreSQL that uses SPTMesh to manipulate and analyze moving objects. SPTMesh and MeshGIS can be used as a framework for future work and investigation.

In this chapter we presented and discussed:

- Our main goals and implementation specific needs.
- The technologies and the architectures that we considered and their advantages and limitations.
- The technologies selected for our implementation.
- The implementation details of SPTMesh and MeshGIS: its dependencies, data structures, operations, architecture, usage and installation and implementation issues.
- A Spatiotemporal Well-Known form for UNIT and MOVING types.
- The structure of the source code of SPTMesh and MeshGIS and the extension of SPTMesh with new triangulation and smoothing methods.
- The use of SPTMesh and MeshGIS as a framework for future work and investigation.

## Data Model Evaluation

In this chapter we present and discuss the tests that were performed to validate the components and the methods that were implemented, and their results.

We used Octave and QGIS to visualize and analyze the tests' results.

### 5.1 Datasets

We used 2 sets of data for testing:

- Synthetic data, i.e., data created manually to test some more or less specific functionality.
- A set of real data obtained from satellite images of 2 icebergs ("RossSea Subsets," 2004) using the methods implemented in (Mesquita, 2013). The icebergs' data corresponds to 9 consecutive time intervals.

### 5.2 Tests

We began by validating our implementation of the triangulation, smoothing and interpolation methods by comparing their results with the results obtained by the original implementation in (Amaral, 2015). For this, we used a specific example and compared the results numerically.

Then, we performed tests with synthetic data, of which we show only a small subset with the more representative and interesting examples, in particular: situations where there are rotations  $\geq 2\pi$  (see Figure 5.1) and the  $180^\circ$  rotation of an object (see Figure 5.2).

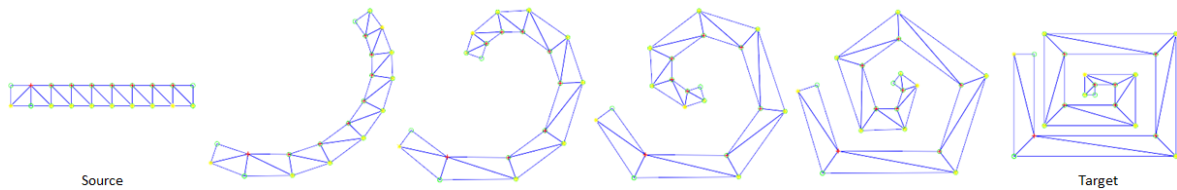


Figure 5.1 - Coil interpolation test.

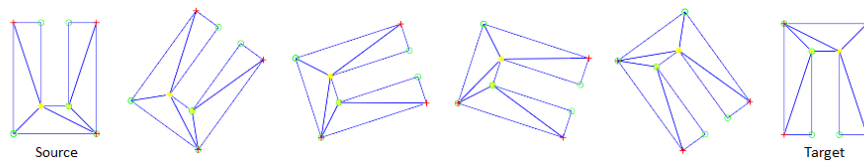


Figure 5.2 -  $180^\circ$  rotation test.

Finally, we performed tests with real data obtained from satellite images of 2 icebergs. See Figure 5.3. The tests were performed using:

- The SPTMesh C++ API directly in a C++ application.
- MeshGIS through SQL in PostgreSQL.

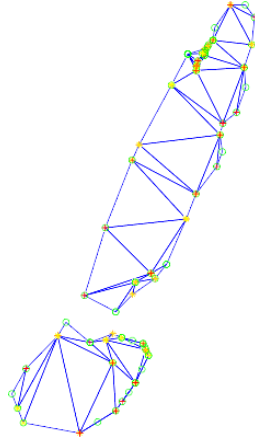


Figure 5.3 - The 2 icebergs used for testing in their initial positions after being processed using SPTMesh.

Figure 5.4, Figure 5.5 and Figure 5.6 show some results, seen in Octave, of the interpolation method. Note that the icebergs are in motion.

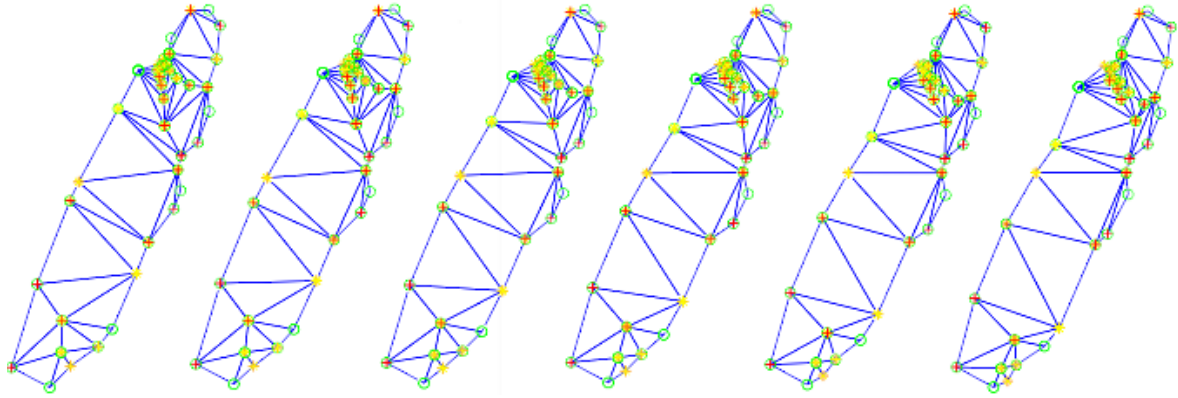


Figure 5.4 - Iceberg 1 interpolation test over an interval of time.

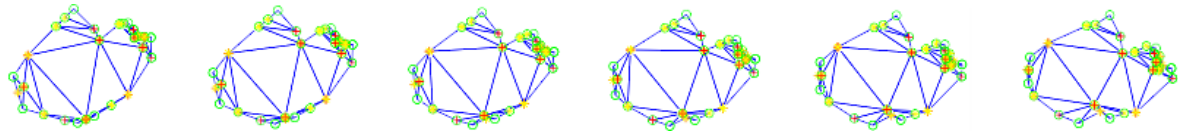


Figure 5.5 - Iceberg 2 interpolation test over an interval of time.

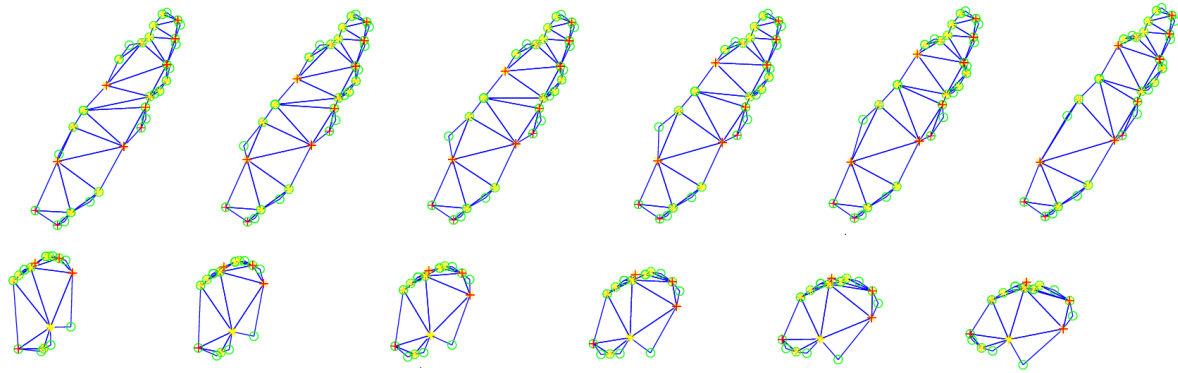


Figure 5.6 - The evolution of the 2 icebergs, seen together, over an interval of time.

To test MeshGIS we start by creating the icebergs table in PostgreSQL where we can store moving regions. For this we use the MovingMesh data type.

```
CREATE TABLE db.icebergs
(
    id        integer,
    name      varchar(50),
    mobj      movingmesh
)
WITH (
    OIDS = FALSE
);
```

We can insert data into the icebergs table using the functions: ST\_MovingMesh\_CreateEmpty and ST\_MovingMesh\_FromSTWKT.

```
INSERT INTO db.icebergs(id, name, mobj) VALUES(1, 'ice 2', ST_MovingMesh_CreateEmpty());

INSERT INTO db.icebergs(id, name, mobj) VALUES(2, 'ice 1'
    ST_MovingMesh_FromSTWKT(
        'MOVINGMESH((1000 2000, (1052 987, ..., 1034 941), (1055 999, ..., 1001 875)))'
    )
);
```

If we search the records in the icebergs table after the 2 previous commands are executed, we obtain the results in Table 5.1. We can see 'ice 1' at instants 1000 and 2000 in Figure 5.7.

```
SELECT * FROM db.icebergs;
```

| <b>Id</b> | <b>Name</b> | <b>Moving Object</b>   |
|-----------|-------------|--|
| 1         | ice 1       | MOVINGMESH((1000 2000, (1052 987, 1090 1037, ..., 1034 941), (1055 999, ..., 1001 875))) |
| 2         | ice 2       | MOVINGMESH EMPTY   |

Table 5.1 - Results of the select command in the icebergs table.

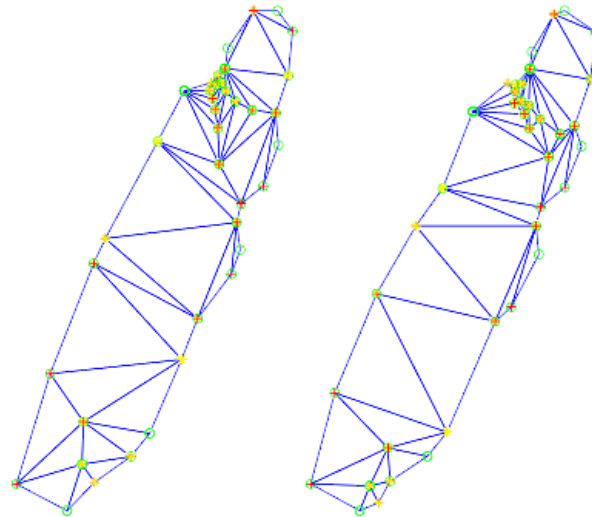


Figure 5.7 - Ice 1 at instants 1000 and 2000.

We can add data to a record in the icebergs table independently of the method used to create it. For example, we can add a unit, i.e., a new evolution of the iceberg during an interval of time, to the record with  $id = 1$ .

```
UPDATE db.icebergs SET mobj = ST_Add_UnitMesh((SELECT mobj FROM db.icebergs WHERE id =
1),
'UNITMESH(2000 3000, (1001 875, 1055 999, 1064.00100000000475 1006.99774999999466, 1073 1015,
1104 1074, 1101 1106, 1106 1127, 1119.50100000000475 1137.997545454487, 1133 1149, 1152 1195,
1144 1218, 1162 1270, 1165.00100000000475 1296.4997735849001, 1168 1323, 1133 1347, 1121 1343,
1096 1301, 1094 1282, 1078 1264, 1069 1266, 1049.50100000000475 1249.99756249999422, 1030 1234,
996 1148, 966 1106, 944.00100000000475 1067.9988421052358, 922 1030, 875 919, 848 817, 903 787,
924 796, 937 820, 979 848), (1030 942, 1078 1043, 1095 1054, 1118 1088, 1130 1114, 1128 1145, 1136
1170, 1146 1171, 1161 1184, 1184 1231, 1177 1254, 1198 1306, 1197 1327, 1207 1357, 1171 1384, 1161
1380, 1135 1348, 1125 1318, 1113 1306, 1102 1309, 1079 1297, 1058 1265, 1027 1194, 979 1138, 932
1052, 908 991, 892 971, 861 869, 919 837, 937 846, 951 869, 996 896))'
, false) WHERE id = 1;
```

After inserting the icebergs' data in the icebergs table we can ask what is: the number of units of the records in the icebergs table and the period in which they are defined. See Table 5.2 and Table 5.3.

```
SELECT ST_get_Size(mobj) FROM db.icebergs;
```

| Number of Units |
|-----------------|
| 9               |
| 9               |

Table 5.2 - Result of the ST\_get\_Size function after the icebergs' data was stored in the icebergs table.

```
SELECT ST_Get_DefTime(mobj) FROM db.icebergs;
```

| Period   |
|--|
| PERIOD(1000 2000, 2000 3000, 3000 4000, 4000 5000, 5000 6000, 6000 7000, 7000 8000, 8000 9000, 9000 10000) |
| ...  |

Table 5.3 - Period in which the records in the icebergs table are defined.



It is also possible to obtain information about the records in the icebergs table in a specific period or instant. See Table 5.4 and Table 5.5, respectively. Figure 5.8 shows ‘ice 2’ at instants 1100, 2000, 3000, 4000 and 4500, respectively. Figure 5.9 shows ‘ice 1’ at instant 1500.

```
SELECT ST_Get_AtPeriod(mobj, 'PERIOD(1100 4500)') FROM db.icebergs;
```

| Moving Object at a Period  |
|--|
| MOVINGMESH((1100 2000, (...), (...)), (2000 3000, (...), (...)), (3000 4000, (...), (...)), (4000 4500, (...), (...))) |
| ...  |

Table 5.4 - Records in the icebergs table defined in a given period.

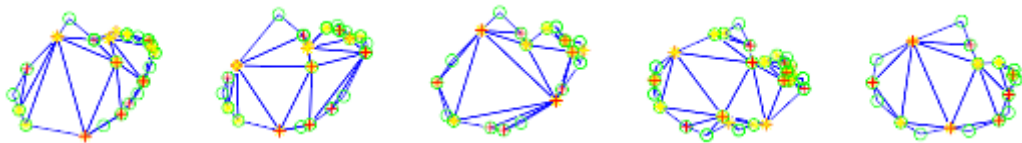


Figure 5.8 - Ice 2 at instants 1100, 2000, 3000, 4000 and 4500.

```
SELECT ST_Get_AtInstant(mobj, 1500) FROM db.icebergs;
```

| Moving Object at Instant 1500                |
|--|
| POLYGON((994.25 909.62, ..., 994.25 909.62)) |
| ...  |

Table 5.5 - Records in the icebergs table defined at a given instant.

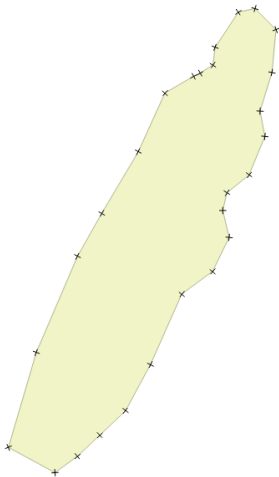


Figure 5.9 - Ice 1 at instant 1500.

Finally, we can search for information about the area, intersection and existence of objects at specific instants or in some cases at specific periods.

Using PostGIS and MeshGIS:

```
SELECT ST_Area(ST_GeomFromText(ST_Get_AtInstant2(mobj, 3500))) FROM db.icebergs;
```

Using only MeshGIS:

```
SELECT ST_Get_Area(mobj, 3500) FROM db.icebergs;
```

We can compare the results from the 2 previous commands in Table 5.6.

| Using             | Area                                 |
|-------------------|--------------------------------------|
| PostGIS + MeshGIS | 66159.5400423294<br>31350.9223673383 |
| MeshGIS           | 66159.5400423298<br>31350.9223673385 |

Table 5.6 - Area of the icebergs in a given instant.

Table 5.7 and Figure 5.10 show the evolution of the area of ‘ice 1’ during a period of time. In Figure 5.10 the red dots correspond to area observations during the specified period and we assume a linear evolution of the area between the observations. In reality, in the general case, this evolution is some non-linear curve.

```
SELECT ST_Get_Area_AtPeriod(mobj, 'PERIOD(1100 10000)') FROM db.icebergs;
```

| Area during the Period (1100 10000)  |
|--|
| MOVINGREAL((1100 2000 67732.8 67389 1), (2000 3000 67389 67907 1), (3000 4000 67720.3 67293 1), (4000 5000 66661.1 67923.5 1), (5000 6000 67252.1 67749.5 1), (6000 7000 67728 65815 1), (7000 8000 65685.1 65307 1), (8000 9000 65344.8 65276.5 1), (9000 10000 64589 64692.5 1)) |
| ...  |

Table 5.7 - Evolution of the area of the icebergs in the table icebergs in a given period.

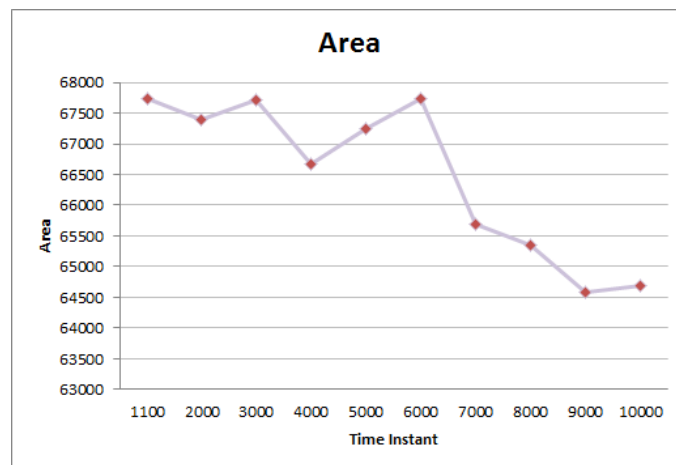


Figure 5.10 - Ice 1 area during the PERIOD(1100 10000).

In the real data set the 2 icebergs do not intersect each other. To test the functions that test the intersection of moving regions, we created 2 moving meshes based on the real data set that intersect each other at some point in a given time interval, Figure 5.11.

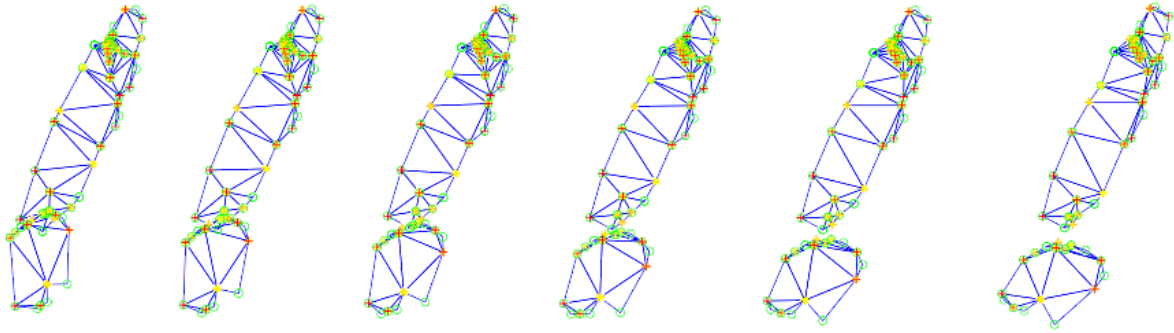


Figure 5.11 - 2 Icebergs intersecting over an interval of time.

Now we can test ST\_Intersect and ST\_Intersection, see Table 5.8, Table 5.9, Table 5.10 and Figure 5.12. The values 1000 and 2000 correspond to the begin and end instants, respectively.

```
SELECT ST_Intersect((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 1000);
```

```
SELECT ST_Intersect((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 2000);
```

| Instant | Intersect |
|---------|-----------|
| 1000    | true      |
| 2000    | false     |

Table 5.8 - ST\_Intersect results.

```
SELECT ST_Intersection((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 1000);
```

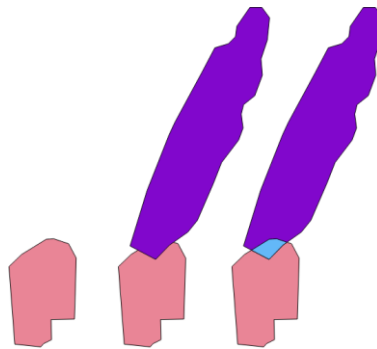


Figure 5.12 - ST\_Intersection result at instant 1000 in light blue, seen in QGIS.

We can use PostGIS to get the area of the intersection between the 2 moving objects at instant 1000. See Table 5.9.

```
SELECT ST_Area(ST_GeomFromText(ST_Intersection((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 1000)));
```

| Intersection Area |
|-------------------|
| 1815.203          |

Table 5.9 - Using PostGIS to get the area of the intersection of 2 moving objects at instant 1000.

```
SELECT ST_Intersection((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 2000);
```

| Intersection             |
|--------------------------|
| GEOMETRYCOLLECTION EMPTY |

Table 5.10 - ST\_Intersection result when no intersection exists.

We can ask if a moving object exists: at an instant, see Table 5.11, and during a period, see Table 5.12.

```
SELECT id, name, ST_Present(mobj, 2100) FROM db.icebergs ORDER BY id;
```

| Id | Name  | Present |
|----|-------|---------|
| 1  | ice 1 | true    |
| 2  | ice 2 | true    |
| 3  | ice 3 | false   |
| 4  | ice 4 | false   |

Table 5.11 - ST\_Present result at instant 2100. Ice 3 and 4 are not defined at that instant.

```
SELECT id, name, ST_Get_Present_AtPeriod(mobj, 'PERIOD(1100 2000, 2500 2750, 3000 4000, 10000 11000)') FROM db.icebergs ORDER BY id;
```

| Id | Name  | Present  |
|----|-------|--|
| 1  | ice 1 | MOVINGBOOL((1100 2000 1), (2500 2750 1), (3000 4000 1), (10000 11000 0)) |
| 2  | ice 2 | MOVINGBOOL((1100 2000 1), (2500 2750 1), (3000 4000 1), (10000 11000 0)) |
| 3  | ice 3 | MOVINGBOOL((1100 2000 1), (2500 2750 0), (3000 4000 0), (10000 11000 0)) |
| 4  | ice 4 | MOVINGBOOL((1100 2000 1), (2500 2750 0), (3000 4000 0), (10000 11000 0)) |

Table 5.12 - ST\_Get\_Present\_AtPeriod result at a given period. (x y 0) means that the object is not defined in the interval [x, y[.

## 5.3 Summary

In this chapter we presented and discussed the tests that we performed and their results and the problems encountered during this phase.

We used 2 data sets for testing:

- Synthetic data, i.e., data created to test more or less specific functionalities, e.g., a geometry coiling.
- A set of real data obtained from satellite images of the evolution of 2 icebergs over time.

We used Octave and QGIS to visualize and analyze the tests' results.

The tests show that our implementation can perform predicate operations, set operations, numeric operations and projection operations on moving objects. They also show that we can use functions from MeshGIS and PostGIS and compose them to provide further value.

We obtained interesting results but there are some problems that have not been solved yet and we discuss them in the next chapter in more detail.



## Discussion

One of the main objectives of this chapter is to discuss the structure of some of the problems that we found and to provide future work guidelines on these issues and some initial background on such a structure.

We present observations, opinions, possible solutions and explanations based on experimentation.

### 6.1 Compatible Triangulation

During the tests phase we realized that the triangulation and interpolation methods were unaware of some of the characteristics of real data, in particular, the presence of cloned vertices, i.e., vertices that have exactly the same x and y coordinates, and collinear vertices, i.e., more than 2 consecutive vertices lying on the same line. The presence of these vertices can make the compatible triangulation method fail. Figure 6.1 shows some examples of cloned and collinear vertices.

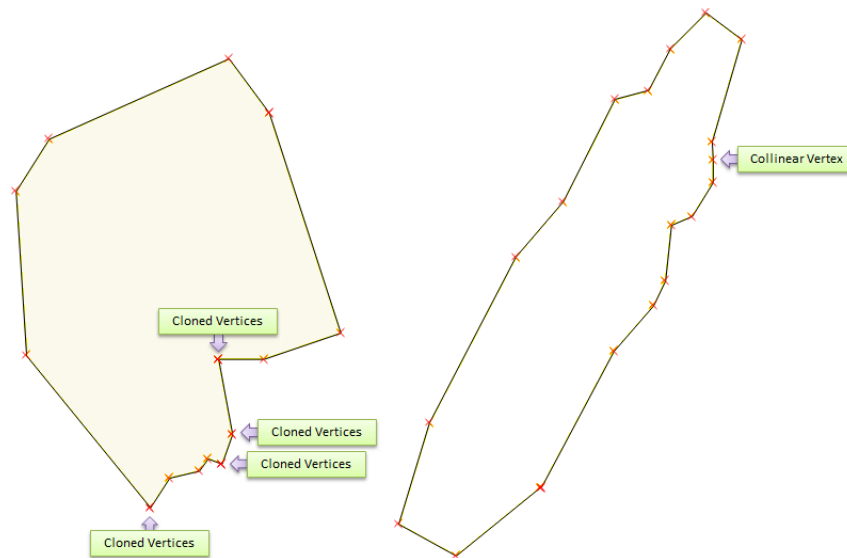


Figure 6.1 - Examples of cloned (left) and collinear (right) vertices.

We tried some quick fixes to solve this problem:

- We allowed triangles to degenerate to a point and to a line. However we found counter-examples where this did not work.
- We avoided cloned vertices by shifting them along the line connecting them to the next non-cloned vertex. This introduces more collinear vertices. Again, this did not work and having collinear vertices seems to increase the probability of failure.

- We avoided cloned vertices as before and collinear vertices by shifting them in the perpendicular line to the line segment where they exist, that passes through them, in both directions, i.e., to the inside and the outside of the polygon. We had cases where we were able to triangulate a polygon by shifting the vertices in both directions, in some cases we had to shift the vertices in a specific direction. This method has the bad property that we are changing the boundary, therefore, introducing bias in the original data. We consider a constant value, say  $\varepsilon_c$ , of 0.001 that was not strictly established, although experiment suggests that smaller values will increase the probability of failure of the triangulation method. But this is not necessarily true because the changes we make in the correspondences have to be seen as a whole. These changes follow the idea that given 3 types of line segments with a cloned vertex on one of its extremities, Figure 6.2, we start by shifting that vertex along the line segment, Figure 6.3. Then we find the line passing through that vertex that is perpendicular to the line segment and use  $\pm\varepsilon_c$  to find the new position of the vertex to the left or to the right of the line segment, Figure 6.4. This idea can be generalized for n cloned or n collinear vertices. As future work, this implementation should be verified and validated.

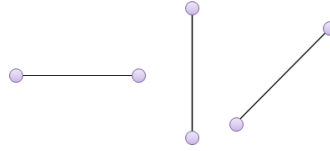


Figure 6.2 - 3 line segments configurations.

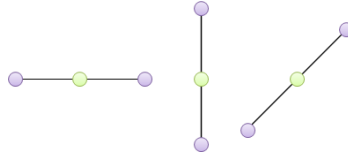


Figure 6.3 - Cloned vertex shifted along the line segments.

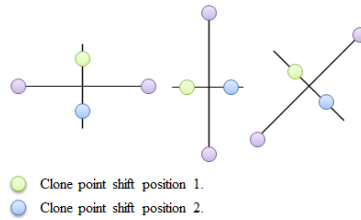


Figure 6.4 - Possible new positions for the cloned vertex shifted along the perpendicular line.

- Finally, we also allowed invalid geometries, i.e., geometries where one or more triangles overlap.

With a proper mixture of these fixes we can triangulate all the examples in our data set of real data. However, this has consequences:

- Different fixes will generate geometries of different complexity, i.e., with more or less Steiner points, and this has a direct impact on the performance of the triangulation and smoothing methods. Such an impact can be noticeable even for relatively simple geometries.



- Invalid geometries, see Figure 6.5, and degenerated triangles can generate problems during interpolation and can cause unexpected rotations under certain conditions.
- Degenerated triangles might cause the SVD of a matrix in the interpolation method to use numeric methods to get approximated results and this has an impact on performance.

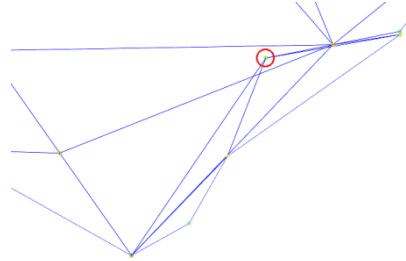


Figure 6.5 - An invalid geometry. Source: smoothing\_test\_ice1\_2\_3\_GEOS\_Clones.m.

The fact that none of the fixes by itself solved the problem suggests that we do not know the structure of the problem and this should be further analyzed:

- How does the probability of failure changes with the number of collinear vertices?
- Do we need to change the triangulation method?
- Do we need to put restrictions on the input of the triangulation method?
- What is the domain of the triangulation method that we are using?
- What is the real structure of the set of geometries that we can have with  $n$  collinear vertices and what is the structure of the set of geometries that we can have when we shift the collinear vertices in the perpendicular line to the line where they are defined?

## 6.2 Interpolation

In this section we discuss the interpolation method problems.

### 6.2.1 Degenerated Triangles and Invalid Geometries

The presence of degenerated triangles and invalid geometries, as discussed in section 6.1, seems to generate unexpected rotations under certain conditions. This is only a claim and it is also possible that this problem may, instead, be related to the unwrap method problem discussed in section 6.2.2.

Figure 6.6 shows the interpolation of a geometry with degenerated triangles and an invalid geometry. This example can be found in the file: interpolation\_test\_ice1\_2\_3\_GEOS\_Clones.m.

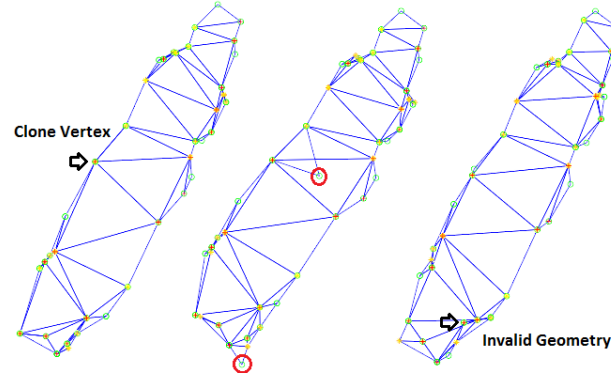


Figure 6.6 - From left to right: source geometry, interpolated geometry at an instant and the target geometry.

The next example seems to indicate that degenerated triangles can in fact generate unexpected rotations. In this example the sequence of the rotation angles was carefully chosen in order to eliminate the unwrap method problem.

Figure 6.7 shows, in red, the evolution of a degenerated triangle (degenerated to a line), more specifically, the evolution of the position of one of the collinear vertices of the degenerated triangle. This example can be found in the file: `interpolation_test_coil_Degenerated.m`. As can be seen there is an unexpected rotation of the degenerated triangle.

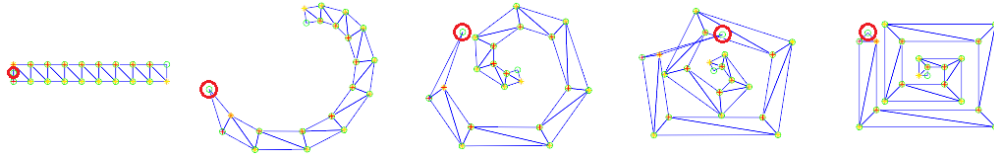


Figure 6.7 - From left to right: source geometry with a degenerated triangle, the interpolated geometry at 3 different instants and the target geometry.

When we eliminate the collinear condition, above, by shifting the vertex in red 0.001 units to the left in the perpendicular line to the line segment where it is defined, and using the same triangles and the same sequence of rotation angles as before, we eliminate the rotation problem.

In Figure 6.8, in red, we can see the evolution of the, now, non-degenerated triangle. This example can be found in the file: `interpolation_test_coil_Degenerated_No_Collinear.m`. This seems to support our claim.

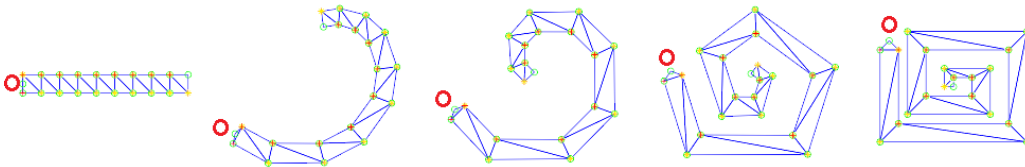


Figure 6.8 - From left to right: source geometry without the degenerated triangle, the interpolated geometry at 3 different instants and the target geometry.

## 6.2.2 Unwrap Method

The original method used to remove discontinuities from the rotation angles of 2 neighbour triangles, i.e., the unwrap method, depends on the sequence of the angles. The method that computes the interpolation components is unaware of such dependency and will generate unexpected triangle rotations.

Let  $A$  be a list of angles,  $\beta_1, \dots, \beta_N$ ,  $|A| > 1$ .

Let  $P$  be the set of all permutations of  $A$  and  $p_i, p_j$  the  $i$ -th and  $j$ -th permutations of  $P$ , respectively.

Let  $E = \forall p_i, p_j : \text{unwrap}(p_i) = \text{unwrap}(p_j)$ .

Then,  $E$  is not a tautology.

An example follows. Given an interval  $I$  and a set  $A$  of triangles' rotation angles, then we can obtain the results shown in Figure 6.9 and Figure 6.10.

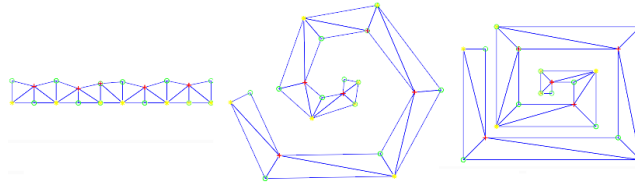


Figure 6.9 - From left to right: instants 1, 8 and 11. Coil interpolation test during an interval  $I$  with a carefully chosen permutation of  $A$ . Source: interpolation\_test\_coil\_tri\_order\_2.m.

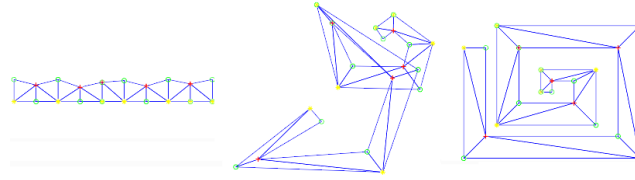


Figure 6.10 - From left to right: instants 1, 8 and 11. Coil interpolation test during an interval  $I$  with a random permutation of  $A$ . Source: interpolation\_test\_coil\_tri\_order.m.

This problem requires further analysis and is not yet solved. Experiment suggests that:

- The sequence of the rotation angles should follow the neighbourhood relationships between the triangles in the geometry, but not every sequence is suitable.
- The problem will occur on geometries with triangles with rotation angles bigger than a  $\beta$ , where  $\beta$  is probably  $\geq 2\pi$ .

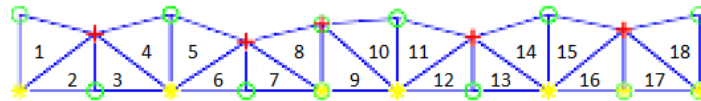


Figure 6.11 - A sequence of triangles with ids from 1, ..., 18.

For the geometry shown in Figure 6.11:

Permutation  $p_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$ , where 1, ..., 18 indicates the id of the triangle for which each element of  $p_i$  stores a rotation angle.

Permutation  $p_j = \{18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$ , where 1, ..., 18 indicates the id of the triangle for which each element of  $p_j$  stores a rotation angle.

Permutation  $p_k = \{9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 2, 3, 4, 5, 6, 7, 8\}$ , where 1, ..., 18 indicates the id of the triangle for which each element of  $p_k$  stores a rotation angle.

Permutations  $p_i$  and  $p_j$  will generate the expected triangles rotations. However,  $p_k$  will not. Finding the ‘ideal’ permutation does not seem to be the ideal solution. What is the correct permutation for more complex geometries like the one in Figure 6.12?

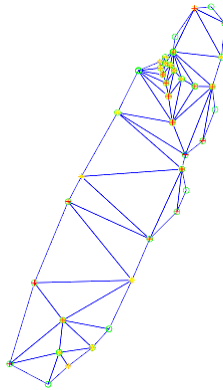


Figure 6.12 - Triangulated geometry of an iceberg.

A possible solution would be to look for discontinuities between pairs of neighbour triangles instead of trying to generate a suitable permutation for all of them?

### 6.2.3 Smoothing Method Relevance

Under certain conditions the use of the smoothing method seems to have influence on the results obtained by the interpolation method. Below, Figure 6.13 and Figure 6.14, we can see the results obtained when we interpolated an iceberg using and not using the smoothing method, respectively. Note that, the first and last meshes have exactly the same boundary on both figures.

In Figure 6.13 we can notice a considerable effect in which there is an increase of the volume of the geometry in the intermediate instants, followed by a decrease of that volume. This growing and shrinking effect is one of the effects that the triangulation and interpolation methods that we use should eliminate.

In Figure 6.14 we can see that there is a considerable deformation of the geometry. This result is both unexpected and undesirable.

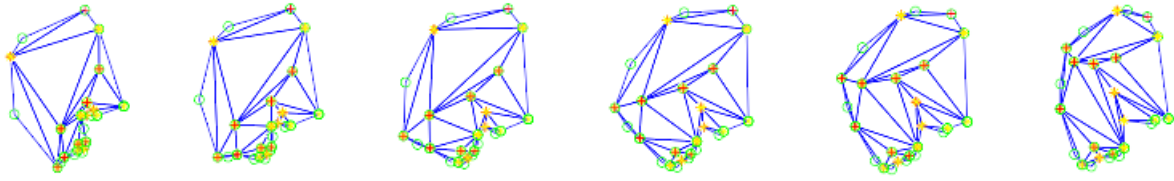


Figure 6.13 - Interpolation using the smoothing method.

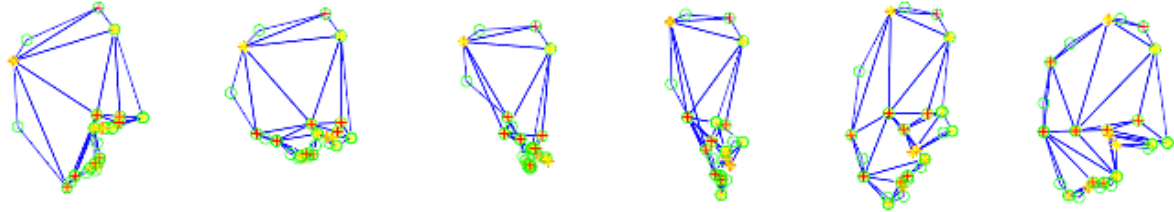


Figure 6.14 - Interpolation not using the smoothing method.

### 6.3 Smoothing Method Performance

The smoothing method has performance issues even when applied to relatively simple meshes, see Figure 6.15. Intuitively, this problem seems to grow with the number of Steiner points. However, it can also be related to the geometry itself.

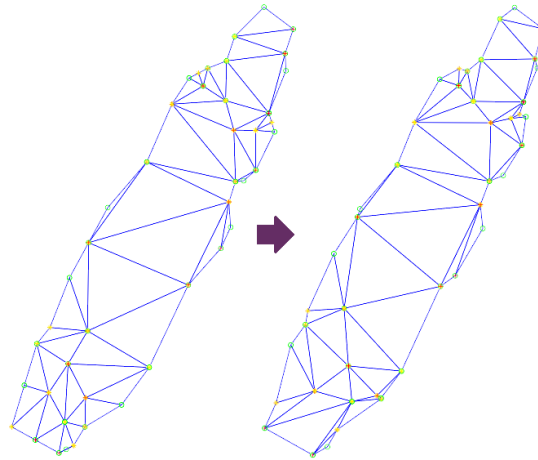


Figure 6.15 - A source and target relatively simple smoothed meshes.

We obtain the results in Table 6.1 when creating the meshes in Figure 6.15 using MeshGIS.

| Methods Used                                | Search Radius | Max Number of Iterations | Smoothing Criteria | Execution Time (sec) |
|---|---------------|--------------------------|--------------------|----------------------|
| Triangulation<br>Interpolation              | 4             | 6                        | Angle              | 1.1                  |
| Triangulation<br>Smoothing<br>Interpolation | 4             | 6                        | Angle              | 34.2                 |

|   |   |   |       |      |
|---|---|---|-------|------|
| Triangulation<br>Interpolation              | 2 | 3 | Angle | 1.1  |
| Triangulation<br>Smoothing<br>Interpolation | 2 | 3 | Angle | 27.3 |

Table 6.1 - Results from a performance test performed in PostgreSQL using and not using the smoothing method.

As we can see in Table 6.1 this is a performance problem that should be considered for future work. The smoothing method allows us to provide the search radius, the maximum number of iterations and the smoothing criteria, i.e., angle or area, to be used. Changing these values can improve performance but we did not test it thoroughly.

## 6.4 Continuity

In this section we discuss continuity issues found during the tests phase.

### 6.4.1 Mesh Objects

The notion of continuity for mesh objects is based, amongst other things, on their positions. The position of a mesh is actually the position of its centroid. While working with real data, we realized that the distance between 2 mesh objects, i.e., the distance between their positions, that are supposed to be continuous, computed using their respective centroids, is some value that can be small or relatively bigger. As a consequence, it is hard to find a value to top up that distance. For this reason, this implementation is not using the mesh position to establish continuity for mesh objects.

Table 6.2 shows the centroid positions, in the 2D Cartesian plane, at the begin and end instants of 9 continuous unit mesh objects, obtained from real data, and their distances computed at the respective continuity instants. The values shown are truncated and they are represented using an abstract metric unit. The centroid computation is only using the boundary points of the respective mesh. The first row represents the distance between the 1° and 2° unit mesh objects, the second the distance between the 2° and the 3° and so on and so forth.

| Centroid at the begin Instant | Centroid at the end Instant | Centroid Distance |
|-------------------------------|-----------------------------|-------------------|
| 1009.17, 1042.75              | 1011.84, 1052.36            | 0.00027           |
| 1011.84, 1052.36              | 1036.94, 1097.33            | 0.25309           |
| 1036.70, 1097.24              | 1059.76, 1134.02            | 1.35299           |
| 1060.62, 1135.06              | 1063.83, 1151.01            | 1.31144           |
| 1064.70, 1151.99              | 1072.77, 1166.77            | 0.46039           |
| 1073.01, 1167.16              | 1106.79, 1194.41            | 0.62456           |
| 1106.43, 1193.91              | 1125.90, 1237.87            | 0.25436           |

|                  |                  |         |
|------------------|------------------|---------|
| 1126.15, 1237.94 | 1126.43, 1248.15 | 2.19076 |
| 1127.99, 1249.69 | 1132.02, 1260.76 |         |

Table 6.2 - Centroid positions, in the 2D Cartesian plane, of 9 continuous unit mesh objects, obtained from real data, and their distances at the continuity instants.

### 6.4.2 Unit Real Objects

When working with real data we realized that it is hard to establish continuity for unit real objects. Unit real objects can represent the evolution of different types of quantities, e.g., the area of a moving object, the perimeter of a moving object and the distance between 2 moving objects. We expect these quantities to be continuous for continuous objects, but the model is not enforcing that. Assuming they are, we need to define a  $\xi$  for each one of these different types of quantities because they have different natures. The problem becomes even harder when we consider the difficulties of finding these  $\xi$  values.

An example follows. Table 6.3 shows the area differences between 9 continuous unit mesh objects, of a moving object representing the evolution of an iceberg over a period of time, computed at the instant where continuity is verified, e.g., 0.027 is the difference between the area of the iceberg at the end and begin instants of the first and second units, respectively. The results obtained seem to indicate that finding a  $\xi$  to establish continuity for the area of a moving object can be somewhat more difficult.

| Area Differences |
|------------------|
| 0.027            |
| 186.746          |
| 631.894          |
| 671.448          |
| 21.500           |
| 129.882          |
| 37.847           |
| 687.485          |

Table 6.3 - The area differences between 9 continuous unit mesh objects.

There is also the possibility that the nature of moving real objects does not impose a notion of continuity on unit real objects. We decided to leave this issue open and in this implementation we do not establish a notion of continuity for unit real objects.

### 6.5 Other Issues

- The use of the STWKT form implies loss of precision. This may have more or less serious consequences. For example, to prevent the triangulation method from failing, we treat cases where there are cloned or collinear vertices. This is done by changing their positions by a small amount. The transformation of an object into the STWKT form can undo these changes due to loss of

precision. As a consequence, it may happen that the triangulation method fails when applied to an object constructed from the new STWKT. And in this case, the process for finding and avoiding collinear vertices becomes more complex. To deal with this situation we use 17-digit precision for the STWKT form, but it is not guaranteed that this will solve the problem for all cases. As future work, we should use a Spatiotemporal Well-Known Binary (STWKB) form instead to avoid loss of precision.

- When adding a unit to a moving object using its STWKT form, the validity of the unit's interval should be verified in the set of intervals of the moving object's units before building the unit. This is particularly interesting and useful for moving mesh objects and that is because constructing a unit mesh object is an expensive process.
- In MeshGIS, when we add or remove a unit to or from a moving object we think it makes sense that the result of these operations clearly indicates its outcome. However, the implementation of this strategy as is involves the transformation of the object resulting from these operations to the STWKT form and then, back to a serialized object. This is unacceptable, in particular for moving mesh objects. When we move from the STWKT form to a serialized object we have to construct all the units of that object, i.e., we have to apply the triangulation, smoothing and interpolation methods. If we imagine a moving mesh object with dozens of units we can get an idea of the consequences of this. The operation to add a unit to a moving mesh no longer uses this strategy.

## 6.6 Summary

We discussed the major problems that were found when working with real data. These problems have not been solved and this discussion serves as a reference for future work and discussion. And although we present observations, opinions, possible solutions and explanations, they are based on experimentation and these problems should be studied and analysed more carefully. Working with real data can potentially help to understand and solve these issues. We also discussed other problems that should be further studied and analysed.



## Conclusions and Future Work

### 7.1 Conclusions

We proposed a discrete data model for moving objects, in particular moving regions, based on the concept of a mesh (a triangulated polygon) and compatible triangulation and rigid interpolation methods. This data model has as its main foundations: the abstract data model presented in (R. H. Güting et al., 2000), the discrete data model proposed in (Forlizzi et al., 2000) and the works in (Mesquita, 2013) and (Amaral, 2015). By using this concept and methods we aim to be able to obtain a more realistic representation of the evolution of moving regions, i.e., a more realistic representation of the changes in position or shape and extent of moving regions over time. To the best of our knowledge this is the first data model for moving objects that uses this concept and methods to represent moving regions. It is not a complete implementation of the abstract data model that it has as a reference. It only considers polygons without holes and a small subset of the operations on moving objects proposed in (Forlizzi et al., 2000). It does not provide a closed system of operations and it does not consider moving lines and moving collections.

We implemented the proposed discrete data model in a C++ library called SPTMesh. It does not implement all the operations defined in the model.

We also implemented a spatiotemporal database extension for PostgreSQL, as a C library, called MeshGIS that uses SPTMesh, as a proof of concept that SPTMesh works with real applications.

SPTMesh is a framework for moving objects. Its main goals are to provide tools to manipulate and analyze moving objects, in particular moving regions. Because SPTMesh is not application specific it can be used by other applications other than MeshGIS.

MeshGIS makes it possible to store the moving objects provided by SPTMesh on PostgreSQL and to manipulate and analyze them using SQL. That is, it makes it possible to take advantage of Database Management Systems (DBMSs) to manage moving objects.

SPTMesh follows the architecture of GEOS a well-known and well-established C++ library to manipulate and analyze spatial objects.

MeshGIS follows the architecture of PostGIS a well-known spatial extension for PostgreSQL.

SPTMesh architecture allows the extension of the library with new triangulation and smoothing methods in a simple way and provides: predicate operations, set operations, numeric operations and projection

operations on moving objects. Furthermore, we can use functions from MeshGIS and PostGIS and compose them to provide further value.

SPTMesh and MeshGIS can be used as a framework for future work and investigation on this area.

We also presented a Spatiotemporal Well-known Text (STWKT) form for moving objects, i.e., a way of expressing moving objects in a ‘standard’ way.

We tested SPTMesh and MeshGIS using the C++ SPTMesh API and the SQL types and functions that bind to the types and functions of MeshGIS. We tested, in particular, the triangulation, smoothing and interpolation methods implemented in SPTMesh. We used 2 data sets for testing: a set of synthetic data and a set of real data describing the evolution of 2 icebergs over time.

During the testing phase we encountered some unexpected problems, e.g., we realized that the triangulation and interpolation methods were unaware of some of the characteristics of the data coming from the method that we use to obtain the real data. We also encountered problems when establishing continuity for the *mesh* and UNIT types. Some of these problems remain unsolved in the current implementation and they are presented and discussed throughout this dissertation and in particular in chapter 6.

We did not perform benchmarking tests but during the testing phase we realized that the smoothing method has performance problems and they are also discussed.

We encountered several implementation problems during this work. The implementation of SPTMesh and MeshGIS started from scratch with little information about the best way to implement them. We used the GEOS and the PostGIS projects as the main references for our implementation. Our main source of information about these projects was its source code. Despite being mature projects with a lot of built in functionality, their source code is not well documented and it was difficult to understand their architectures and structure, i.e., how they work and their main properties. Another problem was the fact that SPTMesh and MeshGIS are implemented using different programming languages, C++ and C. These programming languages have significant differences and in order to implement these components we need knowledge about these 2 programming languages and this is not necessarily a trivial task.

This implementation has limitations but the results obtained are interesting and seem to indicate that the use of the concept of mesh with compatible triangulation and rigid interpolation methods makes some sense and produces promising results.

## 7.2 Main Contributions

We do think that the main contributions of this dissertation are the following:

- We proposed a data model for moving objects that uses concepts and methods that potentially can obtain a more realistic representation of the evolution of moving objects, in particular moving regions. We implemented this model in a framework for moving objects and we obtained results that seem to support this claim.

- We implemented a spatiotemporal extension for PostgreSQL and proved that our framework for moving objects works and it is not application-dependent. MeshGIS proves that: we can store our moving objects in PostgreSQL and manipulate and analyze them using the Structured Query Language (SQL) and we can use functions from MeshGIS and PostGIS and composed them to obtain further value.
- We implemented a framework that, we think, can be used for future work and investigation in this area. With this goal in mind we made an effort to document the classes and their functions with some level of detail in the source code.
- When working with real data we found some problems and we tried to understand their structure. We have a better understanding of ‘the whole’ and of some of the limitations of the methods that we are using.

## 7.3 Future Work

We don’t present a full working framework for moving objects and there is a lot of work ahead. We think that the following issues, not given in any specific order, are major points of interest for future work:

- Solve the problems found for the triangulation and interpolations methods when working with real data. Improve the performance of the smoothing method and establish a solid notion of continuity for the *mesh* and UNIT types.
- Decouple the interpolation method from the UnitMesh class. This is partially done and enables the extension of our framework with different interpolation methods in a simple way. This is interesting since it is not proved that there is a single interpolation method that works for all possible scenarios and it makes it possible to test or integrate other interpolation methods in our framework.
- Provide a Spatiotemporal Well-Known Binary (STWKB) form for MOVING and UNIT types and establish a standard for the STWKT and STWKB forms. This avoids the problems of loss of precision associated with the use of the STWKT form.
- Provide a closed system of operations. We think that having the *mobject* and *uobject* types, i.e., a type that can represent any MOVING type and a type that can represent any UNIT type, respectively, can help to achieve this goal.
- Associate a precision model with the MOVING types, when applicable. And possibly allow the user to define: which one should be used or custom precision models. This is interesting because we want to avoid the loss of precision and its consequences.
- Allow the user to define its own interpolation functions for the *mreal* and *mpoint* types. This makes our framework more powerful and can provide more accurate results for the interpolated values of these types.
- Review the SPTMesh methods so that they provide thread safety or reentrancy properties when relevant.

- Work with dates, locales and time zones. The *instant* type should represent a specific date with a locale and a time zone. We think that it is natural to work with dates independently of its internal representation in SPTMesh.

---

## Bibliography

- Alexa, M., Cohen-Or, D., & Levin, D. (2000). As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00* (pp. 157–164). <http://doi.org/10.1145/344779.344859>
- Amaral, A. (2015). *Representation of spatio-temporal data using Compatible Triangulation and Morphing techniques*. Aveiro University.
- Baxter, W., Barla, P., & Anjyo, K. (2008). Rigid shape interpolation using normal equations. In *NPAR '08 Proceedings of the 6th international symposium on Non-photorealistic animation and rendering* (pp. 59–64). <http://doi.org/http://doi.acm.org/10.1145/1377980.1377993>
- Behr, T., Teixeira de Almeida, V., & Güting, R. H. (2006). Representation of periodic moving objects in databases. *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems - GIS '06*, 43. <http://doi.org/10.1145/1183471.1183480>
- Booth, J., Sistla, P., Wolfson, O., & Cruz, I. F. (2009). A data model for trip planning in multimodal transportation systems. *Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology - EDBT '09*, 994–1005. <http://doi.org/10.1145/1516360.1516474>
- Breunig, M., Turker, C., Bohlen, M. H., Dieker, S., Güting, R. H., Jensen, C. S., ... Scholl, M. (2003). Architectures and Implementations of Spatio-temporal Database Management Systems. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, 263–318. [http://doi.org/10.1007/978-3-642-29066-4\\_{\\_}11](http://doi.org/10.1007/978-3-642-29066-4_{_}11)
- Cotelo Lema, J., Forlizzi, L., Güting, R. H., Nardelli, E., & Schneider, M. (2003). Algorithms for Moving Objects Databases. *The Computer Journal*, 46(6), 680–712. <http://doi.org/10.1093/comjnl/46.6.680>
- Damiani, M. L., & Güting, R. H. (2014). Semantic Trajectories and Beyond, 14–16. <http://doi.org/10.1109/MDM.2014.57>
- Damiani, M. L., Valdes, F., & Güting, R. (2015). Symbolic Trajectories. *ACM Transactions on Spatial Algorithms and Systems*, 1(2), 52. <http://doi.org/https://doi.org/10.1145/2786756>
- de Almeida, V. T., Güting, R. H., Behr, T., Ding, Z., Hoffmann, F., & Spiekermann, M. (2004). SECONDO: An Extensible DBMS Architecture and Prototype. *Informatik-Report 313*. Retrieved from <http://dna.fernuni-hagen.de/papers/Secondo04.pdf>

- Ding, Z., Yang, B., Güting, R. H., & Li, Y. (2015). Network-Matched Trajectory-Based Moving-Object Database : Models and Applications. *IEEE Transactions on Intelligent Transportation Systems*, 16(4), 1918–1928.
- Erwig, M., & Schneider, M. (1999). Developments in spatio-temporal query languages. *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*. <http://doi.org/10.1109/DEXA.1999.795206>
- Erwig, M., & Schneider, M. (2002). Stql - A Spatio-Temporal Query Language. *Kluwer International Series in Engineering and Computer Science*, 105–126. <http://doi.org/citeulike-article-id:4207791>
- Etienne, L., Devogele, T., & Bouju, A. (2010). Spatio-Temporal Trajectory Analysis of Mobile Objects Following the Same Itinerary. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(II), 86–91.
- Forlizzi, L., Güting, R. H., Nardelli, E., & Schneider, M. (2000). A Data Model and Data Structures for Moving Objects Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 319–330). <http://doi.org/http://doi.acm.org/10.1145/342009.335426>
- Gong, C.-Q. (2011). A New Method for the Morph of Planar Polygons Based on Shape Feature. *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, 7–10. <http://doi.org/10.1109/ICM.2011.44>
- Gotsman, C., & Surazhsky, V. (2001). Guaranteed intersection-free polygon morphing. *Computers and Graphics (Pergamon)*, 25(1), 67–75. [http://doi.org/10.1016/S0097-8493\(00\)00108-4](http://doi.org/10.1016/S0097-8493(00)00108-4)
- Gotsman, C., & Surazhsky, V. (2004). *High quality compatible triangulations. Engineering with Computers* (Vol. 20). <http://doi.org/10.1007/s00366-004-0282-6>
- Grumbach, S., Rigaux, P., & Segoufin, L. (2001). Spatio-Temporal Data Handling with Constraints, 95–115.
- Güting, R. H., Almeida, V., Ansorge, D., Behr, T., Ding, Z., Hose, T., ... Telle, U. (2005). SECONDO: An extensible DBMS platform for research prototyping and teaching. *Proceedings - International Conference on Data Engineering*, 1115–1116. <http://doi.org/10.1109/ICDE.2005.129>
- Güting, R. H., Behr, T., & Christian, D. (2012). *Trajectory Databases*. Hagen: Fakultät für Mathematik und Informatik. Retrieved from [https://ub-deposit.fernuni-hagen.de/servlets/MCRFileNodeServlet/mir\\_derivate\\_00000801/Güting\\_Behr\\_Düntgen\\_Trajectory\\_Databases\\_2012.pdf](https://ub-deposit.fernuni-hagen.de/servlets/MCRFileNodeServlet/mir_derivate_00000801/Güting_Behr_Düntgen_Trajectory_Databases_2012.pdf)
- Güting, R. H., Behr, T., & Düntgen, C. (2010). SECONDO : A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 33(2), 56–63.
- Guting, R. H., Bohlen, M., Erwig, M., Jensen, C., Lorentzos, N., Nardelli, E., ... Viqueira, J. (2003). Spatio-temporal models and languages: An approach based on constraints. *Spatio-Temporal Databases*, 177–201. [http://doi.org/10.1007/978-3-540-45081-8\\_4](http://doi.org/10.1007/978-3-540-45081-8_4)

- Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., & Vazirgiannis, M. (2000). A Foundation for Representing and Querying Moving Objects. *ACM Trans. Database Systems*, 25(1), 1–42. <http://doi.org/10.1145/352958.352963>
- Güting, R. H., De Almeida, V. T., & Ding, Z. (2006). Modeling and querying moving objects in networks. *VLDB Journal*, 15(2), 165–190. <http://doi.org/10.1007/s00778-005-0152-x>
- Guting, R., & Lu, J. (2013). Parallel SECONDO Practical and Efficient Mobility Data Processing in the Cloud. In *IEEE International Conference on Big Data* (pp. 17–25).
- Güting, R., & Schneider, M. (2005). *Moving Objects Databases*. (J. Gray, Ed.). Morgan Kaufmann.
- Haesevoets, S., & Kuijpers, B. (2004). Time-dependent affine triangulation of spatio-temporal data. *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems - GIS '04*, 57. <http://doi.org/10.1145/1032222.1032233>
- Hartmut, R., Zhiming, D., & Almeida, V. T. (2006). Modeling and querying moving objects in networks, 15, 165–190. <http://doi.org/10.1007/s00778-005-0152-x>
- Heinz, F., & Güting, R. H. (2016). Robust high-quality interpolation of regions to moving regions. *GeoInformatica*, 20(3), 385–413. <http://doi.org/10.1007/s10707-015-0240-z>
- Hurter, C., Andrienko, G., Andrienko, N., G, R. H., & Sakr, M. (2013). *Air Traffic Analysis. Mobility data: modeling, management, and understanding*. Cambridge University Press.
- Jin, P., & Sun, P. (2008). OSTM: A Spatiotemporal Extension to Oracle. *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*. <http://doi.org/10.1109/NCM.2008.31>
- Jin, P., Yue, L., & Gong, Y. (2005). *Research on a Unified Spatiotemporal Data Model. Symposium on Spatial-temporal Modeling*. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.1303&rep=rep1&type=pdf>
- Kim, D. H., Ryu, K. H., & Kim, H. S. (2000). Spatiotemporal database model and query language. *Journal of Systems and Software*, 55(2), 129–149. [http://doi.org/10.1016/S0164-1212\(00\)00066-2](http://doi.org/10.1016/S0164-1212(00)00066-2)
- Kim, D., Ryu, K., & Park, C. (2002). Design and implementation of spatiotemporal database query processing system. *Journal of Systems and Software*, 60, 37–49. [http://doi.org/10.1016/S0164-1212\(01\)00078-4](http://doi.org/10.1016/S0164-1212(01)00078-4)
- Kim, K.-S., Ogawa, H., Nakamura, A., & Kojima, I. (2014). Sophy: A Morphological Framework for Structuring Geo-referenced Social Media. *Proceedings of the 7th ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, 31–40. <http://doi.org/10.1145/2755492.2755498>
- Matos, L., Moreira, J., & Carvalho, A. (2012). A Spatiotemporal Extension for Dealing with Moving Objects with Extent in Oracle 11G. *SIGAPP Appl. Comput. Rev.*, 12(2), 7–17. <http://doi.org/10.1145/2340416.2340417>

- McKenney, M., Viswanadham, S. C., & Littman, E. (2014). The cmr model of moving regions. *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming*, 62–71. <http://doi.org/10.1145/2676552.2676564>
- Mckenney, M., & Webb, J. (2010). Extracting Moving Regions from Spatial Data. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 438–441). San Jose, California. <http://doi.org/10.1145/1869790.1869856>
- Mckennney, M., & Frye, R. (2015). Generating Moving Regions from Snapshots of Complex Regions. *ACM Trans. Spatial Algorithms Systems*, 1(1), 1–30. <http://doi.org/10.1145/2774220>
- Mesquita, P. (2013). *Morphing Techniques For Representation of Geographical Moving Objects*. Universidade de Aveiro.
- Nidzwetzki, J., & Güting, R. (2016). DISTRIBUTED SECONDO: An extensible highly available and scalable database management system. *INFORMATIK BERICHTE 371 – 05/2016*.
- Noh, S. (2004). Literature Review on Temporal, Spatial, and Spatiotemporal Data Models. *Computer Science Technical Reports. Paper 150*, (April 2000), 1–39. Retrieved from [http://archives.cs.iastate.edu/documents/disk0/00/00/03/50/00000350-00/literature\\_review\(TR04-12\).pdf](http://archives.cs.iastate.edu/documents/disk0/00/00/03/50/00000350-00/literature_review(TR04-12).pdf)
- OGC Moving Features. (2016). OGC Moving Features. Retrieved September 10, 2016, from <http://www.opengeospatial.org/standards/movingfeatures>
- OGC Simple Features. (2016). OGC Simple Feature Access - Part 1: Common Architecture. Retrieved September 10, 2016, from <http://www.opengeospatial.org/standards/sfa>
- Paulo, L. M. (2012). *Morphing techniques in spatiotemporal databases*. Universidade de Aveiro.
- Pelekis, N. (2002). *STAU: A Spatio-Temporal Extension for the Oracle DBMS*. UMIST. Retrieved from <https://books.google.pt/books?id=tHgjcAACA AJ>
- Pelekis, N., Frentzos, E., Giatrakos, N., & Theodoridis, Y. (2008). HERMES: Aggregative LBS via a Trajectory DB Engine. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1255–1258). New York, NY, USA: ACM. <http://doi.org/10.1145/1376616.1376748>
- Pelekis, N., Frentzos, E., Giatrakos, N., & Theodoridis, Y. (2010). *Supporting Movement in ORDBMS – the HERMES MOD Engine*. Hellas.
- Pelekis, N., Frentzos, E., Giatrakos, N., & Theodoridis, Y. (2015). H HERMES : A Trajectory DB Engine for Mobility-Centric Applications. *International Journal of Knowledge-Based Organizations*, 5(2), 19–41. <http://doi.org/10.4018/ijkbo.2015040102>
- Pelekis, N., & Theodoridis, Y. (2006). Boosting location-based services with a moving object database engine. *Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and*



- Mobile Access - MobiDE '06*, 3. <http://doi.org/10.1145/1140104.1140108>
- Pelekis, N., & Theodoridis, Y. (2007). *An Oracle Data Cartridge for Moving Objects*. UNIP-I-SL-TR-2010-01. Hellas. Retrieved from <http://isl.cs.unipi.gr/publications.html>
- Pelekis, N., Theodoridis, Y., Vosinakis, S., & Panayiotopoulos, T. (2006). Hermes - A Framework for Location-Based Data Managment. In *EDBT'06 Proceedings of the 10th international conference on Advances in Database Technology* (pp. 1130–1134). Munich, Germany. [http://doi.org/10.1007/11687238\\_75](http://doi.org/10.1007/11687238_75)
- Pelekis, N., Theodoulidis, B., Kopanakis, I., & Theodoridis, Y. (2004). Literature review of spatio-temporal database models. *The Knowledge Engineering Review*, 19(03), 1–34. <http://doi.org/10.1017/S026988890400013X>
- Popa, I. S., & Zeitouni, K. (2012). Modeling and Querying Mobile Location Sensor Data. In *Proceedings of the 4th Int. Conference on Advanced Geographic Information Systems, Applications, and Services, IARIA* (pp. 222–231).
- Pozzani, G., & Combi, C. (2012). On the semantics of ST4SQL, a multidimensional spatio-temporal query language. In *Proceedings of the 16th International Database Engineering & Applications Symposium on - IDEAS '12* (pp. 222–229). New York, New York, USA: ACM Press. <http://doi.org/10.1145/2351476.2351504>
- Praing, R., & Schneider, M. (2007). Modeling historical and future movements of spatio-temporal objects in moving objects databases. *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management - CIKM '07*, 183. <http://doi.org/10.1145/1321440.1321469>
- Prasad Sistla, A., Wolfson, O., Chamberlain, S., & Dao, S. (1997). Modeling and querying moving objects. In *Proceedings 13th International Conference on Data Engineering* (pp. 422–432). IEEE Comput. Soc. Press. <http://doi.org/10.1109/ICDE.1997.581973>
- RossSea Subsets. (2004). Retrieved September 20, 2016, from <http://rapidfire.sci.gsfc.nasa.gov/imagery/subsets/?project=antarctica&subset=RossSea&date=11/15/20>
- Sanderson, C., & Curtin, R. (2016). Armadillo : a template-based C ++ library for linear algebra. *The Journal of Open Source Software*, 1. <http://doi.org/10.1016/j.cstda.2013.02.005>. Intel.
- Tøssebro, E., & Güting, R. (2001). Creating Representations for Continuously Moving Regions from Observations. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases* (pp. 321–344). Springer-Verlag Berlin Heidelberg. Retrieved from [http://159.226.100.157/sess\\_11293/http182dx.doi.org/10.1007/3-540-47724-1\\_17](http://159.226.100.157/sess_11293/http182dx.doi.org/10.1007/3-540-47724-1_17)
- Tøssebro, E., & Nygård, M. (2011). Representing topological relationships for spatiotemporal objects. *GeoInformatica*, 15(4), 633–661. <http://doi.org/10.1007/s10707-010-0120-5>
- Viqueira, J. R. R., & Lorentzos, N. A. (2007). SQL extension for spatio-temporal data. *VLDB Journal*, 16(2), 179–200. <http://doi.org/10.1007/s00778-005-0161-9>

- Wahid, M. T., Kamruzzaman, A. Z. M., & Shariff, A. R. M. (2006). Spatio-Temporal Object Relational for Biodiversity System (STORe-Biodi). *International Journal of Computer Science and Network Security*, 6(9A), 45–53.
- Xinmin Chen, C., & Zaniolo, C. (2000). SQLST : A Spatio-Temporal Data Model and Query Language. *Conceptual Modeling — ER 2000, 1920*, 111–182. Retrieved from [http://dx.doi.org/10.1007/3-540-45393-8\\_8](http://dx.doi.org/10.1007/3-540-45393-8_8)
- Xu, J. (2012). *Moving Objects with Multiple Transportation Modes*. FernUniversität in Hagen.
- Xu, J., & Güting, R. H. (2012). Manage and query generic moving objects in SECONDO. *Proceedings of the VLDB Endowment*, 5(12), 2002–2005. <http://doi.org/10.14778/2367502.2367558>
- Zhao, L., Jin, P., Zhang, L., Wang, H., & Lin, S. (2011). Developing an Oracle-Based Spatio-Temporal Information Management System. In *International Conference on Database Systems for Advanced Applications* (pp. 168–176). [http://doi.org/https://doi.org/10.1007/978-3-642-20244-5\\_16](http://doi.org/https://doi.org/10.1007/978-3-642-20244-5_16)
- Zheng, Y., Zhang, L., Ma, Z., Xie, X., & Ma, W.-Y. (2011). Recommending friends and locations based on individual location history. *ACM Transactions on the Web*, 5(1), 1–44. <http://doi.org/10.1145/1921591.1921596>